Bounded fairness for probabilistic distributed algorithms*

by

Pepijn Crouzen Ernst Moritz Hahn Holger Hermanns
Saarland University, Saarbrücken, Germany
{crouzen|emh|hermanns}@cs.uni-saarland.de

Abhishek Dhama Oliver Theel
Carl von Ossietzky University Oldenburg, Germany
{abhishek.dhama|theel}@informatik.uni-oldenburg.de

Ralf Wimmer Bettina Braitling Bernd Becker
Albert-Ludwigs-University Freiburg, Germany
{wimmer|braitlin|becker}@informatik.uni-freiburg.de

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog,
Andreas Podelski, Reinhard Wilhelm
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Bounded fairness for probabilistic distributed algorithms[*]

Pepijn Crouzen Ernst Moritz Hahn Holger Hermanns
Saarland University, Saarbrücken, Germany
{crouzen|emh|hermanns}@cs.uni-saarland.de

Abhishek Dhama Oliver Theel
Carl von Ossietzky University Oldenburg, Germany
{abhishek.dhama|theel}@informatik.uni-oldenburg.de

Ralf Wimmer Bettina Braitling Bernd Becker
Albert-Ludwigs-University Freiburg, Germany
{wimmer|braitlin|becker}@informatik.uni-freiburg.de

April 26, 2010

Abstract

This paper investigates quantitative dependability metrics for distributed algorithms operating in the presence of sporadic or frequently occurring faults. In particular, we investigate necessary revisions of traditional fairness assumptions in order to arrive at useful metrics, without adding hidden assumptions that may obfuscate their validity. We formulate distributed algorithms as Markov decision processes to incorporate both probabilistic faults and non-determinism arising from the distributed setting. We especially discuss the notions of bounded fairness and near-round-robin schedulers, which appear particularly suited for distributed algorithms running on nearly symmetric infrastructure, as it is common for sensor network applications. We further develop methods to incorporate this fairness notion in the quantitative model checking of distributed algorithms. Finally, we apply our methodology to several case studies to provide a first experimental insight into its applicability.

1 Introduction

In a distributed algorithm several separated processes with only local knowledge must cooperate to achieve a common goal. Such algorithms have been extensively studied over more than 40 years [23]. Abstract properties of distributed algorithms are traditionally established by proving correctness guarantees under certain assumptions.

Such analyses may be called *qualitative* as they establish a particular quality of the algorithm under study. It is interesting to note that in this type of analysis we usually see strong assumptions about the dependability of the system. Either it is (implicitly) assumed that no communication fault occurs [23], or it is assumed that only a limited number of faults occur [17], or it is assumed that at some point faults stop occurring.

Example 1. *As a running example we use a distributed self-stabilizing minimal spanning tree (MST) algorithm, which is a simplification of the one in [16]. Given a set of processes $V = \{1, \ldots, N\}$ which are connected in a weighted graph (V, E) with one special process called the* root, *the task of the algorithm is to compute, for each process, its distance to the root along edges of the graph. For*

[*]This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS) (see www.avacs.org for more information) and the Graduiertenkolleg "Leistungsgarantien für Rechnersysteme"

an edge $(i,j) \in E$ we write $d_{i,j} \in \mathbb{N}$ for its weight, which represents the fixed distance of process i to process j. Each node knows only the distance to its direct neighbours. We write c_i for the current root distance estimates of process i. The estimates c_i are initially set to an arbitrary value. Each process i now perpetually repeats the following updates:

$$c_i := \begin{cases} 0, & \textit{if } i \textit{ is the root,} \\ \min\{d_{i,j} + c_j \mid (i,j) \in E\}, & \textit{otherwise.} \end{cases}$$

Notably, this algorithm relies on a dependable mechanism for process i to inspect the current values c_j. Assuming absence of any fault, it provably converges to the setting where each variable c_i indeed holds the distance of process i to the root [16]. Throughout this paper we will investigate what properties we can show for this algorithm in the presence of transient probabilistic faults.

With the rise of the internet and of wireless networks, solutions capable of operating in unpredictable and unreliable environments have found practical relevance. For the field of distributed algorithms this means that we can no longer make strong assumptions about the absence of faults. Especially sensor networks must operate in an environment where the frequent occurrence of faults is the rule, not the exception.

Recently there has been an effort to quantitatively analyze distributed algorithms [27, 14, 22, 21]. Instead of proving that an algorithm works under a set of assumptions, *quantitative* analysis aims at studying *how well* an algorithm works, under some assumptions.

This shift in focus implies that established assumptions, models and algorithms that have been used for qualitative analysis are not necessarily adequate in the quantitative setting. First of all we must alter the assumptions on dependability. Instead of assuming that there only occur a limited number of faults, or that faults occur only in a certain time-period, we now assume that transient faults can occur at any time, but with a specific probability. This allows us to calculate the probability that the algorithm works correctly [22, 21] if fault probabilities are given.

This paper focuses on a further set of assumptions that must be reconsidered in order to facilitate insightful quantitative analysis. These assumptions concern the order in which the different processes operate in a distributed algorithm. For a qualitative analysis, it is common to pose the most general assumption under which the algorithm is assumed to work correctly. Dijkstra, for instance, states that "nothing may be assumed about the relative speeds of the N computers" [23] in his seminal work on distributed mutual exclusion.

However under such an assumption quantitative analysis quickly becomes useless: the worst-case success probability (that all processes can enter their critical section within some finite time) is provably zero under such a weak assumption. Also standard fairness assumptions do not change this phenomenon, as we will discuss.

More realistic assumptions have also been studied in the context of qualitative analysis, where the amount of time between two steps of a process is *bounded* [9, 19, 29]. This paper investigates the concept of bounded fairness in the context of quantitative analysis of probabilistic distributed algorithms and presents the following contributions:

1. We formally define the notion of bounded fairness for probabilistic models with non-determinism.

2. We show that if we wish to realistically address the relative speeds of processes in a distributed algorithm under the assumption of transient probabilistic faults, we must consider non-deterministic models.

3. We provide two possible approaches to this modeling problem, which are tailored to models running on symmetric infrastructure and prove that they indeed capture the notion of bounded fairness for probabilistic and non-deterministic models.

4. We show how to analyze dependability aspects of a distributed algorithm when considering non-deterministic models.

5. We further show the effectiveness of our methodology by applying it to several case studies.

The models we are dealing with are non-deterministic and probabilistic at the same time, and known as Markov decision processes [43]. As our modeling and analysis vehicle, we use the probabilistic model checker PRISM [36], and its modeling language.

The work presented here draws some inspiration from a recent comparative scheduler study [10]. It can be considered in sharp contrast to the simulation-based analysis of distributed systems as it prevails in the systems community. In that approach, dependability metrics are often skewed by hidden assumptions that are added by the simulator, the libraries, the runtime environment, the random number generator, or the postprocessing of simulation traces. In the wireless sensor community, there is a growing awareness [4, 31, 35] for this problem, and this paper constitutes a step towards a solution.

2 Preliminaries

In this section we briefly discuss the general theory of Markov decision processes and guarded command languages, specifically the one used by the PRISM tool [36].

2.1 Markov decision processes

We denote the collection of all probability distributions on (Borel) subsets of a set S as $\mathcal{P}(S)$.

Definition 1. *A Markov decision process (MDP) is a tuple (S, A, R) where: S is the state space, A is a set of actions and $R : S \times A \times \mathcal{P}(S)$ is the transition relation such that for every pair $(s, a) \in S \times A$ we have at most one transition $(s, a, \pi) \in R$. If we find* exactly *one transition (s, a, π) per pair (s, a), we say that the MDP is* enabled.

The state space and the set of actions are discrete. For the remainder of this section we fix a MDP $M = (S, A, R)$.

We say that an action $a \in A$ is *enabled* in a state $s \in S$ if there exists a triple (s, a, π) in R. We denote the set of enabled actions in a state $s \in S$ as A_s. For an enabled MDP we have that $A_s = A$ for all states $s \in S$. We write $\bar{R}(s, a)$ for the random variable which takes values in S and is distributed according to π. A *path* of M is a, possibly infinite, sequence of states and actions $\sigma = s_1, a_1, s_2, a_2, \ldots$, where each action a_i is enabled in the previous state s_i. A finite path σ ends in state $last(\sigma)$. The length of a path is equal to the number of actions in the path. This means that a path consisting of a single state has length 0. We denote the set of all finite paths as $(S \times A)^* \times S$ and the set of all infinite paths as $(S \times A)^\omega$. Given a path $\sigma = s_1, a_1, s_2, a_2, \ldots$ the *action trace* $\sigma[A]$ of σ consists of the sequence of actions in σ, i.e. $\sigma[A] = a_1, a_2, \ldots$.

A *scheduler* f is a function from finite paths to distributions over the actions $f : ((S \times A)^* \times S) \to \mathcal{P}(A)$ such that for any finite path $\sigma = s_1, a_1, \ldots, s$ we have for $f(\sigma) = \pi$ that, if the distribution π assigns a probability greater than zero to a, then a is enabled in s. We write $\bar{f}(\sigma)$ for the random variable which takes values in A and is distributed according to π. A scheduler f induces a Markov chain (MC) $(X^k_{f,M})_{k \in \mathbb{Z}_+}$ with state space $(S \times A)^* \times S$ where for finite paths $\sigma = s_1, a_1, \ldots, s$ and $\sigma' = s_1, a_1, \ldots, s, a, s'$ (i.e., σ is a largest prefix of σ') we have:

$$P(X^{k+1}_{f,M} = \sigma' | X^k_{f,M} = \sigma) = P(\bar{f}(\sigma) = a) \cdot P(\bar{R}(s, a) = s')$$

All other transition probabilities are zero, since they are not selected by the scheduler under consideration. If we now fix a starting distribution over S, we can compute the probability of MDP M being in a state $s \in S$ at a time-point $k \in \mathbb{Z}_+$ given a scheduler f by summing over all paths of length k that end in s.

A scheduler is called *deterministic* if it always chooses a point distribution over the set of actions, i.e., a distribution where one action has probability 1 and the others have probability 0. We refer to a set of schedulers as a *scheduler class*. The set of all schedulers for a MDP M is denoted C_M.

In the analysis of MDPs, it is often considered interesting to study the maximum or minimum probability of being in a state $s \in S$ in the Markov chains $X_{f,M}$ induced by all schedulers f in a scheduler class. We refer to these probabilities as extremal probabilities. Extremal probabilities correspond to best-case and worst-case behaviors. Thus, bounds on minimal and maximal probabilities imply guarantees for any possible behavior of the system. For instance, we can use these bounds to prove safety properties of the system. If we wish to show that a system is in a "safe" state at a certain time-point with at least probability p, we need only show that the minimum probability to be in this "safe" state is larger than p.

Definition 2. *Given a MDP $M = (S, A, R)$, a set of initial states $S_{init} \subset S$, a scheduler class C, a set of goal-states $G \subset S$ and a time-point $k \in \mathbb{Z}_{\geq 0}$ we define the infimum* transient probability *function $\mathcal{R}^-$ as follows:*

$$\mathcal{R}^-(M, S_{init}, C, G, k) = \inf_{f \in C, s \in S_{init}} P(X_{f,M}^{(k)} \in G \mid X_{f,M}^{(0)} = s)$$

The supremum transient probability *function $\mathcal{R}^+$ is obtained by replacing* inf *by* sup *in the above.*

The infimum long-run average probability *function $\mathcal{S}^-$ is the Cesàro limit of the infimum transient probability:*

$$\mathcal{S}^-(M, S_{init}, C, G) = \lim_{t \to \infty} \frac{1}{t} \sum_{k=0}^{t-1} \mathcal{R}^-(M, S_{init}, C, G, k).$$

Again the supremum long-run average probability *function $\mathcal{S}^+$ is defined analogously.*

For finite models, the Cesàro limits exist and match the standard steady-state limits, i.e., $\lim_{k\to\infty} \mathcal{R}^-(M, S_{init}, C, G, k)$ and $\lim_{k\to\infty} \mathcal{R}^+(M, S_{init}, C, G, k)$, if these limits exist [43]. The extremal transient probability for a time-point k ranging over all schedulers is always attained by a deterministic scheduler with step-bound k.

2.2 Extremal probabilities of scheduler classes

We now consider the problem of computing extremal probabilities for a subset C of all schedulers. The algorithms mentioned in Subsection 2.1 find extremal probabilities over *all* schedulers and can, in general, not be used to compute extremal probabilities over a strict subset C of all schedulers, since the scheduler that realizes the extremal probability may not belong to C. We show in this subsection that for certain classes of schedulers, relevant to our analysis goal, we can construct a new MDP M' such that the extremal probabilities in M over the schedulers in C agree with the extremal probabilities over all schedulers in M'. This then allows to apply the aforementioned algorithms on M', resulting in the extremal probabilities for scheduler class C of M.

Definition 3. *Given a MDP $M = (S, A, R)$ and a function F from finite paths to sets of actions, $F : ((S \times A)^* \times S) \to 2^A$ such that for a path σ ending in state s we have that $F(\sigma)$ contains only actions enabled in s, $F(\sigma) \subset A_s$. Class C_F is the set of all schedulers f of M which satisfy the following: for all paths σ of length k which are reached with non-zero probability by f, this scheduler only selects actions contained in $F(\sigma)$ with non-zero probability. Formally:*

$$P(X_{f,M}^{(k)} = \sigma) > 0 \wedge P(\bar{f}(\sigma) = a) > 0 \Rightarrow a \in F(\sigma).$$

We say that the function F characterizes *the scheduler class C_F.*

We now show that for any MDP M and scheduler class C_F there exists another MDP M' such that the extremal probabilities over C_F for M can be derived from the extremal probabilities over all schedulers for M'.

Theorem 1. *Given a MDP $M = (S, A, R)$, a set of initial states $S_{init} \subset S$ and a scheduler class C_F characterized by a function F, there exists a MDP $M' = (S', A', R')$ and functions $g : S' \to S$ and $h : S \to S'$ such that, for any set $G \subset S$ and time-point k, we have:*

$$\mathcal{R}^-(M, S_{init}, C_F, G, k) = \mathcal{R}^-(M', h(S_{init}), C_{M'}, g(G), k)$$
$$\mathcal{S}^-(M, S_{init}, C_F, G) = \mathcal{S}^-(M', h(S_{init}), C_{M'}, g(G))$$

where $h(S_{init}) = \{h(s) \mid s \in S_{init}\}$ and $g(G) = \{x \mid x \in S' \wedge g(x) \in G\}$. Furthermore, the supremum probabilities for M and M' are related in the same way as the infimum probabilities.

The proof of Theorem 1 can be found in Appendix A.

2.3 Guarded commands

We use the PRISM guarded command language (GCL) to model distributed algorithms. A *program* consists of concurrently running *modules* and a predicate `init`, which denotes the initial states. Each of the modules contains a number of *variables* and *guarded commands*. Variable declarations consist of a name and a range. Guarded commands are of the form $[a]$ $\mathtt{g} \to \mathtt{p}_1 : \mathtt{X'}\texttt{=}E_1 \texttt{ + } \ldots \texttt{ + } \mathtt{p}_k : \mathtt{X'}\texttt{=}E_k$. Here a is an action name, `g` is a guard, which is a predicate over program variables, and $\mathtt{X'} = E_1, ..., \mathtt{X'} = E_k$ are assignments to the variables of the superordinate module, weighted with probabilities $p_1, ..., p_k$ where $\sum_{i=1}^{k} p_i = 1$.

The semantics of a program is an induced MDP $M = (S, A, R)$. A state s of M is an evaluation of the program variables. The actions A are the action names occurring in the guarded commands of the source program. The set of initial states are the states fulfilling `init`.

Relation R is derived from the guarded commands: assume s fulfills the guard of the command given above. Then there is a transition (s, a, π) in R. The distribution π is such that for each probabilistic choice $i \in [1, k]$ we have $P(\bar{R}(s, a) = s_i) = \sum_{j \in [1,k] \wedge s_i = s_j} p_j$ where s_m is the state obtained by applying the assignment $\mathtt{X'}\texttt{=}E_m$ to s. Variables to which no value is assigned remain unchanged. By default, only one guarded command enabled in some module of the program is executed at once. This is different however, if an action name a occurs in commands of several modules. The execution of any of those commands requires that in each module the guard of at least one command labeled with a is satisfied. These commands are then executed synchronously in all participating modules.

3 Modeling assumptions

In the following subsections we discuss what assumptions we make on the models of distributed algorithms regarding communication and faults. We will also show how such models can be specified in a guarded command language in general and in particular for our running example. With these assumptions we try to find a balance between a model that is still feasible to analyze and a model that reflects realistic distributed algorithms.

3.1 Communication model

In a distributed algorithm there are multiple processes, each with only partial (local) knowledge. Information is exchanged through communication infrastructure and this is what enables the processes to still solve global problems. There are two communication model paradigms predominant in the literature, namely the message passing model and the shared memory model. We give an overview of different ways of implementing and modeling communication in Section 6. For reasons of simplicity we employ a variation of the shared memory approach.

We say every process in a distributed algorithm performs *local* algorithmic steps and has a *local* state space. This is in contrast to the *global* steps of the combined model and the *global* state space. We say that processes communicate through links and we say that processes that are linked are *neighbors*. We now make the following assumptions in the context of communication and distributed algorithms:

1. Local steps of a process only change the local state.

2. Local steps are deterministic. Although we could easily support non-determistic algorithms, we choose to consider only deterministic local algorithms to focus on the non-determinism arising from the distributed setting.

3. A process can always perform a local step, even if this has no effect. We include this assumption to simplify fairness discussions as we do not have to consider whether a process is enabled or not.

4. Local steps are interleaved. That is, we do not consider the possibility of two local steps in two different processes to happen at exactly the same time. This greatly simplifies our models and our fairness notions. This assumption is also justified by the fact that we mainly consider distributed settings where different processes operate with different, inaccurate, local clocks.

5. Local steps only depend upon the local state of a process and the state of neighboring processes. In this way we model communication between processes. This simple form of communication simplifies the modeling task and reflects communication through message buffers. It may be useful to restrict the knowledge a process has about its neighbors, for instance we may restrict it to a special communication register. To formalize such a restriction the notion of partial observability [41] can be used, but this is not considered here.

We now discuss how the MDP model of a distributed algorithm as described above can be modeled in the GCL. We associate to each process one module and one unique action. Each command in a module is labeled with the associated action. In the underlying MDP the actions now specify which local process is responsible for each global transition.

Inherited from PRISM, variables are already partitioned into a set of local variables for each module. Each module can only update its local variables, which ensures the model satisfies assumption 1 above. By ensuring that the guards in a module are pairwise disjoint (i.e., the conjunction of any pair of guards is always equivalent to `false`) we ensure that it is impossible that two commands in the same module are enabled at the same time. This in turn ensures that the local algorithms are deterministic (assumption 2). By ensuring that the disjunction of all guards is equivalent to true, we achieve that at all times at least one guard is enabled, which means assumption 3 holds. Since we use different actions to label the local steps of different processes we have no synchronization, which means local steps of different processes will interleave (assumption 4). Finally we must make sure that the guards and right hand sides of actions in a module contain only local variables or variables of modules of neighboring processes to ensure assumption 5 holds.

Example 2. *In Table 1 we show how to model our running example of the self-stabilizing minimal spanning tree algorithm. The graph under consideration consists of four nodes and all connections have length one. The modules for nodes three and four are derived from the module of node two by renaming variables and action names. Since PRISM does not support infinite variable ranges, here and in the following, we abstract register values larger than the maximal path length by a value of the maximal path length plus one, which is* 4 *here.*

3.2 Fault model

Given the nature of distributed algorithms, where many simple components cooperate to achieve a common goal, we have to deal with the possibility of faults occurring. Instead of complicating the design of the distributed components to protect against faults, distributed algorithms themselves are designed to tolerate such faults. We give a brief account of different fault models seen in the literature in Section 6. Again we list our assumptions on faults and show how to incorporate such faults in a GCL model.

We make the following assumptions.

```
mdp

module node1 // node 1 is the root node
  n1 : [0..4];
  [a1] (true) -> n1' = 0;
endmodule

module node2 // connected to nodes 1 and 4
  n2 : [0..4];
  [a2] (true) -> n2'= min(n1+1,n4+1,4);
endmodule

// node 3 is connected to nodes 2 and 4
module node3=node2[n2=n3,n1=n2,n4=n4,a2=a3]
endmodule

// node 4 is connected to nodes 3 and 2
module node4=node2[n2=n4,n1=n2,n4=n2,a2=a4]
endmodule
```

Table 1: MST algorithm on a four node graph.

1. Faults are local. This means every fault occurs in one of the processes and only affects this process. The alternative notion, that a fault may change multiple processes is difficult (though not impossible) to model as we would need to use synchronization to affect the different changes in different modules. The notion of local faults arises naturally from the fact that we consider distributed algorithms where local processes run on separate hardware. Still, we could employ global faults to account for the possibility of multiple local faults occurring within one step of the algorithm.

2. Effects of faults are caused by or linked to activity. This means that the impact of a fault in a process only appears when this process executes a local algorithm step.

3. Faults are probabilistic. We assume that for each local step taken by a process there is some prespecified probability that a fault occurs. This assumption enables us to establish properties of the algorithm even though faults continue to occur. For pragmatic reasons, the effect of a fault is also assumed to be probabilistic. For instance, a fault may change the value of a state variable to a value chosen randomly with a uniform distribution over all possibilities. Another option is to allow *Byzantine* faults, where the effect of a fault is not specified (i.e., it is non-deterministic). In that context, the effect of a fault can be studied in *worst-case* or *best-case* scenarios. We choose to consider fully probabilistic faults, as Byzantine faults greatly increase the amount of non-determinstic branching, which is as yet detrimental to the effectiveness and performance of our analysis techniques.

We model faults in the GCL model of a distributed algorithm by refining it using probabilism. Transitions now have several possible updates, each with its own probability. For brevity, we illustrate how to extend our running example with communication garbling errors.

Example 3. *Table 2 shows how to include communication garbling errors in our running example. When communication is garbled, the active node reads a random value for the shortest distance variable of one of its neighbors instead of the actual value. The probability of communication garbling is defined as a constant. Note that it is not possible for a non-root node i to set its distance variable n_i to zero after a message gets garbled. This is because the least value the nodes can receive as the shortest distance of any neighbor is zero. This would then cause the node to set its own distance variable to one.*

```
const garble = 0.1;
const no_garble = 1 - garble;
...
module node2 // connected to nodes 1 and 4
  n2 : [0..4];
  [a2] (true) ->
    nogarble : (n2'= min(n1+1,n4+1,4)) +
    garble / 4 : (n2'=1) + garble / 4 : (n2'=2) +
    garble / 4 : (n2'=3) + garble / 4 : (n2'=4);
endmodule
...
```

Table 2: MST algorithm with communication garbling.

4 Scheduling of independent processes

Given that in a distributed system no single process has global control, it is not clear in which order the different processes execute steps of their local algorithms. Even if considering symmetric hardware with identical clock speed settings for all partners, the order of execution cannot be assumed fixed, due to the unavoidable phenomenon of clock drift.

Thus, the order of execution must be considered to be *non-deterministic*. We now consider what this means for the worst-case stabilization time of a self-stabilizing algorithm, in absence of any assumption on the order of execution.

Example 4. *Assume that, at the start of the minimal spanning tree algorithm, none of the processes has the correct distance value stored locally. A possible order of execution is one where a single process continuously executes local steps. It is obvious that for this order we will never reach a desirable state in which all nodes know their distance to the root even in the absence of transient faults. This is because other processes are blocked indefinitely.*

This well-known phenomenon is usually addressed by some fairness assumptions. We extend now the notion of bounded fairness, which restricts the time that is allowed between two local steps in a process, to MCs induced from the MDP model of a distributed algorithm.

4.1 Period of a scheduler

We wish to reason about the relative speeds of the processes in a distributed algorithm. We quantify the speed of a process v by counting how many other processes execute steps between two subsequent steps of process v. In the MDP models of distributed algorithms each process v is associated with a unique action a_v. Now given a scheduler for such an MDP, we look at the paths induced by this scheduler and the distances between consecutive a_v-transitions. We call this distance the *period* of the path at a particular time-point.

Example 5. *Consider the MDP (S, A, R) of a distributed algorithm with 4 nodes such that $A = \{a, b, c, d\}$. Now we look at an example of a path of the MDP:*

$$s_1, b, s_2, c, s_3, a, s_4, b, s_5, d, s_6, c, s_7, a, \ldots.$$

In particular we are interested in the action trace of the path and the number of steps until the current action appears again, i.e. the period.

action:	b	c	a	b	d	c	a	...
period:	3	4	4	...				

The period tells us at what intervals the nodes in the distributed algorithm are capable of executing local steps. Since we consider stochastic models we define the period of an MDP given a particular scheduler as a stochastic process which tells us the probability of observing a particular period at a particular time-point.

Definition 4. *Given an MDP $M = (S, A, R)$, with a scheduler f that induces a MC $X_{f,M}$, the* period *of the induced MC at time-point k is a stochastic process $\left(\Lambda^{(k)}_{f,M}\right)_{k \in \mathbb{Z}_{\geq 0}}$ which takes values in $\mathbb{Z}_{>0} \cup \{\infty\}$ and has distribution:*

$$\begin{aligned} P(\Lambda^{(k)}_{f,M} = i) = \sum_{a \in A} & P(\bar{f}(X^{(k)}_{f,M}) = a \\ & \wedge \bar{f}(X^{(k+1)}_{f,M}) \neq a \wedge \ldots \wedge \bar{f}(X^{(k+i-1)}_{f,M}) \neq a \\ & \wedge \bar{f}(X^{(k+i)}_{f,M}) = a), i \in \mathbb{Z}_{>0}, \\ P(\Lambda^{(k)}_{f,M} = \infty) = 1 - & \sum_{i \in \mathbb{Z}_{>0}} P(\Lambda^{(k)}_{f,M} = i). \end{aligned}$$

Note that $\bar{f}(X^{(k)}_{f,M})$ is the decision of scheduler f at time-point k and $P(\bar{f}(X^{(k)}_{f,M}) = a)$ is then the probability that scheduler f selects action a at time-point k. We write $\Lambda^{(k)}$ when the MDP and the scheduler are clear from the context.

If we now consider all possible schedulers for an MDP M then, for the worst-case expected period length we have for any $k \in \mathbb{Z}_{\geq 0}$ that $\sup_{f \in C_M} E(\Lambda^{(k)}_{f,M}) = \infty$. This worst-case scheduler is any scheduler that schedules an action a at time k with probability one, but schedules that same action with probability zero for all subsequent steps.

The period of an induced Markov chain does not tell us anything about the *first occurrence* of a particular action. To be able to enforce that the first occurence of an action is not delayed indefinitely we define it as a random variable.

Definition 5. *Given an MDP $M = (S, A, R)$, with a scheduler f that induces a MC $X_{f,M}$ and an action $a \in A$ the* first occurence *of a, $\Delta^{(a)}_{f,M}$ is a random variable which takes values in $\mathbb{Z}_{>0} \cup \{\infty\}$ and has distribution:*

$$\begin{aligned} & P(\Delta^{(a)}_{f,M} = i) = \\ & \qquad P(\bar{f}(X^{(0)}_{f,M}) \neq a \wedge \ldots \wedge \bar{f}(X^{(i-2)}_{f,M}) \neq a \\ & \qquad \wedge \bar{f}(X^{(i-1)}_{f,M}) = a), i \in \mathbb{Z}_{>0}, \\ & P(\Delta^{(a)}_{f,M} = \infty) = 1 - \sum_{i \in \mathbb{Z}_{>0}} P(\Delta^{(a)}_{f,M} = i). \end{aligned}$$

The notion of periodicity allows us to reason about bounded fairness in the context of probabilistic models in a similar way as has been done for purely non-deterministic models [9]. We can say an induced MC is bounded fair if its period (and first occurences) lies within certain bounds with probability one. In Subsection 4.3 we will extend bounded fairness to models with non-determinism *and* probabilistic transitions. We will define bounded fairness for MDPs by finding the class of schedulers that induces exactly the set of MCs whose period is bounded with probability one. Before we do so, we review some pragmatic solutions to the problem of blocking described in Example 4.

4.2 Pragmatic solutions

A different, more pragmatic, approach to fairness problems is to consider specific instances, especially by fixing a particular scheduler and then analyzing the induced model [10, 21]. Two different kinds of schedulers are used widely: *round-robin* schedulers, which fix a particular execution order for the processes and force the distributed algorithm to adhere to this order, and the *randomized*

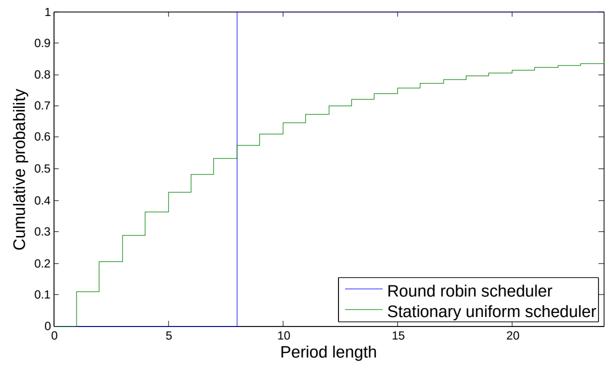

Figure 1: Cumulative distribution of the period $\Lambda^{(k)}$ for a round-robin and the stationary uniform scheduler.

scheduler, which assigns equal probabilities to each enabled action for every path. These types of schedulers also occur in the form of hidden assumptions in simulation environments for distributed algorithms [4, 34, 15].

Definition 6. *For an enabled MDP $M = (S, A, R)$ of a distributed algorithm with N processes, the* stationary uniform*, or randomized, scheduler f for M picks each action $a \in A$ with equal probability, for all paths σ: $P(f_R(\sigma) = a) = 1/N$.*

Scheduler f for M is a round-robin *scheduler if a bijection $g : A \to [0, N-1]$ exists which defines an order on the actions of M such that for any finite path σ of length k and any action $a \in A$ we have:*

$$P(f(\sigma) = a) = \begin{cases} 1, & \text{if } k \bmod N = g(a) \\ 0, & \text{otherwise.} \end{cases}$$

For a round-robin scheduler we find $\Lambda^{(k)} = N$ for all time-points k. While this is well suited for symmetric hardware systems without drifting clocks, we consider this a too strict assumption for a distributed algorithm. Since perfect clocks are unrealistic one would need a reliable clock synchronization service on which the algorithm runs. Though this is feasible, it is expensive, and limits the application domain of distributed algorithms unnecessarily even for safety-critical applications.

For a stationary uniform scheduler we find that $\Lambda^{(k)}$ is geometrically distributed with parameter $1/N$ for all time-points k. We feel that there is no good reason to assume that the period of a distributed algorithm is geometrically distributed. For example the variance of a geometrically distributed random variable with parameter $1/N$ is $N^2 - N$, which is highly unrealistic in practical applications. Figure 1 compares the distribution of the period $\Lambda^{(k)}$ for round-robin and stationary uniform schedulers. We can see from this figure that the round-robin schedulers are completely deterministic with respect to period length, while the stationary uniform scheduler allows components to act multiple times in a row or not at all for a long period of time, with some probability. The advantage of using a stationary uniform scheduler is that it is memoryless and therefore easy to analyze or simulate.

4.3 Scheduler classes

We now introduce two new types of scheduler classes which are aimed at realistically capturing the possible schedules for distributed algorithms running on nearly symmetric infrastructures. The first scheduler class extends the notion of bounded fairness with lower and upper bounds to MDPs. The second scheduler class uses the notion of rounds to restrict computations. This scheduler class leads to an over-approximation of bounded fairness, but can be analyzed more efficiently.

4.3.1 Bounded fairness revisited

We extend the notion of bounded fairness by restricting the schedulers such that the period length of the induced MC is bounded both from above and from below. Given a path of length k, $\sigma = s_1 a_1 s_2 a_2 \dots s_k a_k s_{k+1}$ we write $A_\sigma^{(i)}$ for the set of all actions that occur in the last i steps of the path if $i \leq k$. In case $i > k$, $A_\sigma^{(i)}$ is undefined:

$$A_\sigma^{(i)} = \begin{cases} \{a_j \mid k - i < j \leq k\}, & \text{if } i \leq k \\ \textit{undefined}, & \text{if } i > k. \end{cases}$$

Definition 7. *Given an enabled MDP M with $|A| = N$ and bounds $L, U \in \mathbb{Z}$ with $1 \leq L \leq N$ and $N \leq U$. We define the class $C_{[L,U]}$ of $[L, U]$* bounded fair *schedulers characterized by the function $F : ((S \times A)^* \times S) \to 2^A$, which describes which actions are allowed for any finite path $\sigma \in ((S \times A)^k \times S)$ of length $k \in \mathbb{Z}_{\geq 0}$.*

$$F(\sigma) = \begin{cases} A \setminus A_\sigma^{(U-1)}, & \textit{if } k \geq U \wedge |A \setminus A_\sigma^{(U-1)}| = 1 \\ A \setminus A_\sigma^{(k)}, & \textit{if } k < L \textit{ or} \\ & k < U \wedge |A \setminus A_\sigma^{(k)}| = U - k \\ A \setminus A_\sigma^{(L-1)}, & \textit{otherwise.} \end{cases}$$

The intuitive meaning of function F defined in Definition 7 is as follows. For paths longer than $U - 1$ we have that, if a particular action did not occur in the last $U - 1$ steps, then we must pick this action to ensure the period does not exceed U (first case). If there is no such action, we must pick an action that has not appeared in the last $L - 1$ steps to ensure the period is at least L (last case).

For paths shorter than L we must pick an action that has not occurred yet (second case, first condition); this avoids that the first period is less than L. Finally we have for paths of length $k < U$ that, if the number of actions that have not yet occurred equals $U - k$, then we must schedule one of these missing actions (second case, second condition). This last requirement ensures that the situation will never occur that multiple actions have not occurred in the last $U - 1$ steps. We now give an example of how the function F works to ensure bounded fairness.

Example 6. *Consider the enabled MDP $M = (S, A, R)$ of a distributed algorithm with 4 processes, and action set $A = \{a, b, c, d\}$. Below we list the action traces of several example paths and the corresponding sets of actions allowed by the function F, which characterizes $[2, 5]$ bounded fairness as defined in Definition 7.*

$\sigma[A]$	$F(\sigma)$	$\sigma[A]$	$F(\sigma)$
ϵ	$\{a, b, c, d\}$	$abac$	$\{d\}$
a	$\{b, c, d\}$	$abacd$	$\{a, b\}$
ab	$\{a, c, d\}$	$abacda$	$\{b\}$
aba	$\{c, d\}$	$abacdab$	$\{c, d\}$

We can see that for this particular path we have that the first three period lengths are 2, 5, and 3.

The following theorem shows the relationship between $[L, U]$ bounded fairness and the periodicity of the MDP.

Theorem 2. *Given an enabled MDP $M = (S, A, R)$ with $|A| = N$ and given bounds $L, U \in \mathbb{Z}$ with $1 \leq L \leq N$ and $N \leq U$, a scheduler f is $[L, U]$ bounded fair if and only if for all time-points $k \in \mathbb{Z}_{\geq 0}$ the period of the induced MC at time-point k lies between L and U:*

$$P\left(\Lambda_{f,M}^{(k)} \in [L, U]\right) = 1,$$

And for any action a the first occurrence of a is less than or equal to U:

$$P\left(\Delta_{f,M}^{(a)} \leq U\right) = 1.$$

The proof of Theorem 2 can be found in Appendix B.

In Subsection 4.4.1 we show how we can use the function F from Definition 7 to compute extremal probabilities for the class of $[L, U]$ bounded fair schedulers.

4.3.2 Round-based schedulers

In the previous subsection we have seen how we can restrict the observed period lengths for an MDP by using $[L, U]$ bounded fairness. In this subsection we take a different approach. Instead of choosing a distribution over the actions for each path we divide paths into *rounds*, in which each action is taken exactly once, and choose a distribution over the possible orders for such a round. For the sake of brevity we give only an informal discussion of round-based schedulers, although they can be defined in a similar way as the general definition of schedulers.

Recall that a round-robin scheduler is determined by a bijection from the actions to the integers between 1 and N, where N is the number of actions. This function describes the order in which the actions should occur (see Definition 6). *Randomized round-robin* (RRR) schedulers are schedulers that give, for each round, a distribution over the round-robin orders depending on the path followed so far. The scheduler then selects actions deterministically in the order it has chosen for that round.

Consider two subsequent round-robin orders r and r' for an MDP with N actions. For some action a we may have $r(1) = a$ and $r'(N) = a$. Then the distance between the two executions of action a is exactly $2N - 1$. For the case $r(N) = a$ and $r'(1) = a$ we find that the distance is only 1. It is then clear that any RRR-scheduler is $[1, 2N - 1]$ bounded fair (Note that the reverse is not true). We now introduce a class of schedulers which we conjecture is $[N - k, N + k]$ bounded fair.

A *k-restrictedly randomized round-robin* (k-RRRR) scheduler is a RRR scheduler that may only pick certain RR orders in each round, based on the RR order used in the previous round. Consider a path σ where the last round of N actions were taken according to the RR order r. Now a k-RRRR scheduler may only schedule a RR order r' with probability greater than zero if the following holds. For any action $a \in A$ we have that the distance between the position of a in r and the position of a in r' is at most k.

Example 7. *Consider an enabled MDP $M = (S, A, R)$ with $A = \{a, b, c, f\}$ and consider a path σ of length divisible by 4 (i.e., a path consisting of a number or rounds) where the last N actions occurred according to the RR order bacd. Now we have that a 1-RRRR scheduler may only schedule the following RR orders with probability greater than zero: bacd, abcd, bcad, badc, and abdc. The RR order acbd is, for instance, not allowed since action b goes from position 1 to position 3 and the difference in position then exceeds 1.*

Now consider two subsequent round-robin orders r and r' for an MDP with N actions, such that the distance between the position of an action a in r is i and the position of a in r' is between $i - k$ and $i + k$ for some $0 \leq k < N$. It is clear that the distance between the two executions of a lies between $N - k$ and $N + k$. In general, we conjecture that any k-RRRR-scheduler is a member of the class of $[N - k, N + k]$ bounded fair schedulers. This then allows us to use the extremal probabilities over the class k-RRRR schedulers as conservative estimates of the extremal probabilities for the $[N - k, N + k]$ bounded fair schedulers.

In Subsection 4.4.2 we show how to use GCL encodings to compute extremal probabilities for the class of k-RRRR schedulers.

4.4 Computing extremal probabilities for scheduler classes

In this section we show how to compute extremal transient reachability probabilities and extremal steady-state probabilities classes of $[L, U]$ bounded fair schedulers and k-RRRR schedulers. In most cases we cannot apply the known algorithms directly to our MDP model because the standard algorithms calculate the extremal probabilities for all schedulers. However, we use the result of Theorem 1 to construct a new MDP that we can use to calculate the desired metrics. This new MDP is constructed by adding a module to the original PRISM model that restricts actions that are available depending on the path followed so far.

4.4.1 Enforcing $[L, U]$ bounded fair schedulers

We fix an enabled MDP $M = (S, A, R)$ with $A = \{a_1, \ldots, a_N\}$ such that $|A| = N$. We now take a closer look at the definition of the function F in Definition 7, which prescribes which actions are allowed for the class of $[L, U]$ bounded fair schedulers. We now consider what information about the current path σ is actually necessary to decide which actions are allowed. We can see that for each action $a_i \in A$ we must sometimes know whether a_i is in one of the sets $A \setminus A_\sigma^{(U-1)}$, $A \setminus A_\sigma^{(k)}$, and $A \setminus A_\sigma^{(L-1)}$, where $k < R$. To establish this it is enough to know how many steps ago in σ each action a_i occurred. We denote this distance $d(\sigma, a_i)$ which is defined recursively, where $s \in S$ and $a_j \in A$:

$$d(\sigma, a_i) = \begin{cases} \textit{undefined}, & \text{if } \sigma = s \\ d(\sigma', a_i) + 1, & \text{if } \sigma = \sigma' a_j s \wedge a_i \neq a_j \\ 1, & \text{if } \sigma = \sigma' a_j s \wedge a_i = a_j \end{cases}$$

We now have that $a_i \in A \setminus A_\sigma^{(k)}$ if and only if $d(\sigma, a_i) \leq k$. Since we consider only $[L, U]$ bounded fair schedulers all paths σ for which we have $d(\sigma, a_i) > U$ occur with probability zero. Finally we must also take care of the corner cases where the length of path σ is less than L or less than U.

To implement this scheduler class, we add an additional PRISM module *scheduler* to the original model of M which records the values of $d(\sigma, a_i)$ and path-length k. Care must be taken since PRISM only allows bounded integers. The scheduler module keeps track of the values of d for each action and the current path and the value of k. It then uses these variables to restrict the actions that are allowed given the current path. This restriction is done through *synchronization.* The commands in the module for each local process v has one action name associated with it, which represents the action $a_v \in A$ in the MDP. The scheduler module now has one command per process module labeled with the same action name. The guard of this command is true exactly when the action a_v is in $F(\sigma)$. In this way the scheduler module makes sure that only bounded fair schedules are followed.

Example 8. *Table 3 shows the scheduler module for the class of* $[3, 5]$ *bounded fair schedulers for an enabled MDP with four actions. The bounds are denoted by the constants L and* U*. The variables* c_i *represent the values* $d(\sigma, a_i)$ *for* $1 \leq i \leq N$*. The formulas* x_i*,* y_i*, and* z_i *are used to represent whether* $a_i \in A \setminus A_\sigma^{(R-1)}$*,* $a_i \in A \setminus A_\sigma^{(j)}$*, and* $a_i \in A \setminus A_\sigma^{(L-1)}$ *respectively for current path* σ *of length* j*. The formulas* $count_x$ *and* $count_y$ *represent* $|A \setminus A_\sigma^{(R-1)}|$ *and* $|A \setminus A_\sigma^{(k)}|$ *respectively. The formulas are shown in Table 4.*

4.4.2 Enforcing k-RRRR schedulers

As for the class of k-RRRR schedulers, we also add a scheduler module to the PRISM specification of MDP M which restricts the possible options for the schedulers.

The scheduler module keeps track of the current round-robin order r. This order determines which actions are chosen in the current round. The module further keeps track of the current position in the round j, i.e. $j = l \bmod N$ where l is the length of the current path. The scheduler module enforces at all times that the correct action is taken according to the current round-robin order. This means an action a_i is only enabled if action a_i has position j in the order r, i.e. $r(a_i) = j$.

For the case that $j = N - 1$ a k-RRRR scheduler may change the round-robin order, but only such that for all actions the distance between the positions of the action in the current and the

```
const int L=3;
const int U=5;

module scheduler
  k : [0..U] init 0;
  c1 : [0..U-1] init 0;
  c2 : [0..U-1] init 0;
  c3 : [0..U-1] init 0;
  c4 : [0..U-1] init 0;

  [a1] (((k=U) & (count_x=1) & x0) |
    ((k<L) & y0) |
    ((k<U) & (count_y=U-k) & y0) |
    ((
      ((k=U) & !(count_x=1)) |
      ((k<U) & !(count_y=U-k))
    ) & z0)) & (c2<U-1) & (c3<U-1) & (c4<U-1) ->
    (c1'=0) & (c1'=c2+1) & (c2'=c3+1)
      & (c3'=c4+1) & (k' = (k<U ? k+1 : k));
  [a2] ...
  [a3] ...
  [a4] ...
endmodule
```

Table 3: The scheduler module which restricts the allowed schedulers to the $[3, 5]$ bounded fair schedulers where the MDP has actions $A = \{a_1, a_2, a_3, a_4\}$.

```
formula x0 = (c0 >= U-1);
formula x1 = (c1 >= U-1);
...
formula y0 = (c0 >= k);
...
formula z0 = (c0 >= L-1);
...
formula count_x = ((x0 ? 1 : 0) + (x1 ? 1 : 0)
    + (x2 ? 1 : 0) + (x3 ? 1 : 0));
formula count_y = ((y0 ? 1 : 0) + (y1 ? 1 : 0)
    + (y2 ? 1 : 0) + (y3 ? 1 : 0));
```

Table 4: The formulas used by the PRISM module in Table 3.

next round is less than or equal to k. So, when $j = N - 1$ several commands for each action will be enabled, each selecting a different allowable round-robin order for the next round.

Example 9. *Table 5 shows an example of a scheduler module for the class of k-RRRR schedulers for an MDP with N actions where $k = 1$ and $N = 4$. The module distinguishes between the case where $j < N - 1$ and the order is not changed and the case where $j = N - 1$ where a new ordered is picked.*

The formulas p_i, with $1 \leq i \leq N$ denote whether a particular action is allowed given the values of j and r. The formulas q_i with $1 \leq i \leq N!$ denote whether it is allowed to select the i-th round-robin order based on the current round-robin order r. Note that the round-robin orders are lexicographically ordered (for this example we have that the first order is $a_1a_2a_3a_4$ and the last order is $a_4a_3a_2a_1$). Table 6 shows some of the formulas p_i and q_i.

We have developed a simple C program which automatically generates the GCL code for the scheduler module for the class of k-RRRR schedulers, given the number of actions $|A|$ and k.

5 Case studies

In this section we illustrate the use of bounded fairness for distributed algorithms with probabilistic faults by applying our approach to three case studies. For each algorithm we compute bounds for the probability that the system is in a *safe* configuration, i.e., its maximum and minimum *availability*. We consider transient availabilities (the probability to be in a safe state at a certain time-point) and long-run average availabilities (the expected percentage of time to be in a safe configuration if

```
module scheduler
  r : [1..M];
  j : [0..N-1] init 0;

  [a1] (j<N-1) & p1 -> (j'=j+1);
  [a2] (j<N-1) & p2 -> (j'=j+1);
  ...

  [a1] (j=N-1) & p1 & q1 -> (j'=0) & (r'=1);
  [a1] (j=N-1) & p1 & q2 -> (j'=0) & (r'=2);
  ...
  [a2] (j=N-1) & p2 & q1 -> (j'=0) & (r'=1);
  ...
  [a3] (j=N-1) & p3 & q1 -> (j'=0) & (r'=1);
  ...
  [a4] (j=N-1) & p4 & q1 -> (j'=0) & (r'=1);
  ...
endmodule
```

Table 5: The scheduler module which restricts the allowed schedulers to the 1-RRRR schedulers where the MDP has actions $A = \{a_1, a_2, a_3, a_4\}$.

```
formula p1 =
  ((j=0) & ((r=1)| (r=2)| (r=3)|
    (r=4)| (r=5)| (r=6))) |
  ((j=1) & ((r=7)| (r=8)| (r=13)|
    (r=14)| (r=19)| (r=20))) |
  ((j=2) & ((r=9)| (r=11)| (r=15)|
    (r=17)| (r=21)| (r=23))) |
  ((j=3) & ((r=10)| (r=12)| (r=16)|
    (r=18)| (r=22)| (r=24)));
...

formula q1 = ((r=1)| (r=2)| (r=3)| (r=7)| (r=8));
formula q2 = ((r=1)| (r=2)| (r=5)| (r=7)| (r=8));
...
formula q24 = ((r=17)| (r=18)| (r=22)| (r=23)| (r=24));
```

Table 6: The formulas used by the PRISM module in Table 5.

the system runs forever). Since we are interested in studying the effect of non-determinism arising from the interleaving of processes we have fixed, for each case study, a single starting state. We have used the PRISM model-checker [30] to compute transient availabilities and a semi-symbolic algorithm to compute long-run average availabilities [48]. All PRISM models are available upon request.

5.1 Minimal spanning tree algorithm

We consider the distributed minimal spanning tree algorithm (MST) developed throughout the paper. We use a fault model where communication may be garbled (with probability 0.1) or may fail to take place (with probability 0.2). When a node fails to communicate with another (i.e., it is not able to read the value of the distance variable of one of its neighbours), then it simply does not change its state. We consider here a network of four nodes. In the starting state we have that all nodes have their distance parameter set to the maximum value.

In Figure 2 we give best- and worst-case transient *availabilities*, i.e., the figure shows the probability to be in a state in which all processes have correctly computed their distance to the root node at a certain time-point. We consider two different $[L, U]$ bounded fair (BF) scheduler classes, the class of all round-robin (RR) schedulers and the stationary uniform (SU) scheduler, for which we find the unique transient availability directly, instead of a maximum and minimum.

In the figure we can see the effect of increased scheduler freedom. For each scheduler class, transient availability is initially low, but increases as the nodes exchange information. While the class of RR schedulers is rather restricted, since the nodes always execute in the same order, bounded fairness allows some nodes to act faster than others. In the worst case, nodes that have not

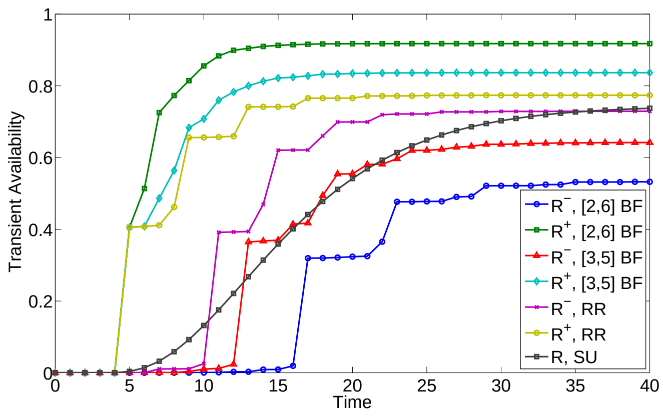

Figure 2: Transient availabilities for the MST algorithm. Minimum and maximum availabilities are shown for two classes of bounded fair (BF) schedulers and the class of round-robin (RR) schedulers. The transient availability for the stationary uniform (SU) scheduler is also shown.

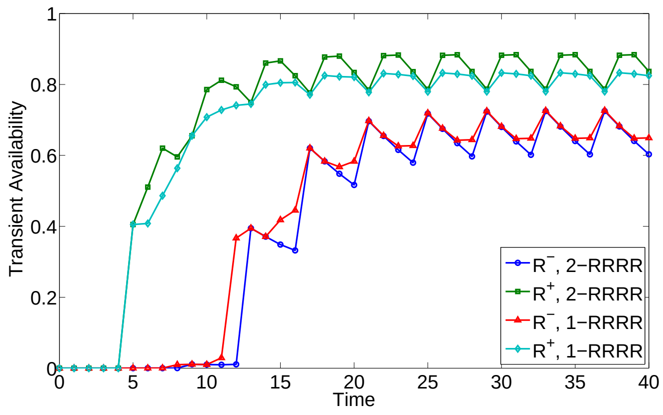

Figure 3: Transient availabilities for the MST algorithm. Minimum and maximum availabilities are shown for two classes of k-RRRR schedulers.

yet computed the correct distance from the root are scheduled last, while nodes that have already correctly established their distance from the root are scheduled as soon as possible. We see that this has scheduling freedom has a noticeable effect on the bounds for the transient availabilities.

The transient availabilities of the stationary uniform scheduler show us that this is *not* a good choice of scheduler. Its transient availability alternatively lies well below or well above the minimal bounds of all the other scheduler classes. Most importantly, the effect that the algorithm needs several “rounds” to establish stability can be clearly seen for the class of RR schedulers and the classes of BF schedulers by “jumps” in the transient availability curves. For the SU scheduler this important effect cannot be seen as its curve is smooth.

Figure 3 shows the transient availabilities for both the class of 1-RRRR and 2-RRRR schedulers. As expected, the bounds given by the 1-RRRR class stay within the bounds of the $[3, 5]$ bounded fair class and similar for the 2-RRRR and $[2, 6]$ bounded fair classes. We see that the RRRR schedulers display a round-based behavior. For the first action within a round, these classes give the most freedom of choice, which is reflected in the more extreme bounds for this time-point. For the last action within a round, the round-based schedulers have no freedom of choice; they must choose the one action that has not yet occurred within the round. As a result we see that there is very little difference in maximal and minimum transient availability for the associated time-points.

The long-run average availabilities for the different scheduler classes can be seen in Table 7. The results clearly show that with increased scheduling freedom, the bounds for the long-run average availability become less tight.

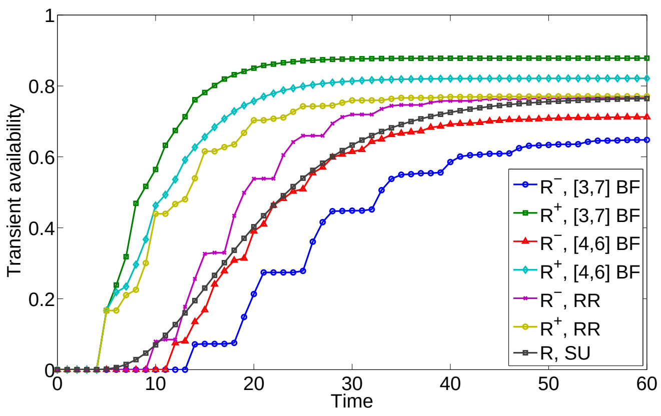

Figure 4: Transient availabilities for the gossiping algorithm. Minimum and maximum availabilities are shown for two classes of bounded fair (BF) schedulers and the class of round-robin (RR) schedulers. The transient availability for the stationary uniform (SU) scheduler is also shown.

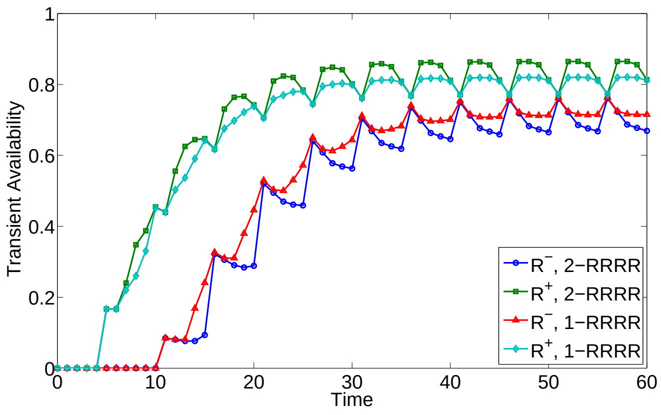

Figure 5: Transient availabilities for the gossiping algorithm. Minimum and maximum availabilities are shown for two classes of k-RRRR schedulers.

5.2 Gossiping information spread algorithm

Our second case study considers a gossiping information spread algorithm inspired by [18]. We again have a set of distributed processes organized in a graph. The processes work together to distribute information throughout the network. One process initially posesses information, while the other processes randomly read from their neighbors to try to obtain this information. A process not having the information can aquire it by reading from a neighbor which has. We consider a network with five nodes.

We consider communcation faults and local faults. A communication between two nodes fails with probability 0.2 and as a result the state of the active process does not change. In the case of a local fault (which occurs with probability 0.05 whenever a process executes a local step) a process loses the information it has. The process that initially has the information can never lose it.

Figure 4 depicts the probability that the information is known by all processes at a certain time-point, i.e., the transient availability. The behavior of availabilities is similar to that for the MST case study. Again we see that when we consider bounded fair schedulers, as opposed to round-robin schedulers, the minimum availability of the system is significantly decreased, meaning that the system is vulnerable to clock-drift. As for the first case study, we observe that the stationary uniform scheduler does not accurately reflect the system behavior.

Figure 3 shows the transient availabilities for both the class of 1-RRRR and 2-RRRR schedulers. Similar observations as in the case of the MST algorithm can be made. The long-run average availabilities for the gossiping case study can also be found in Table 7.

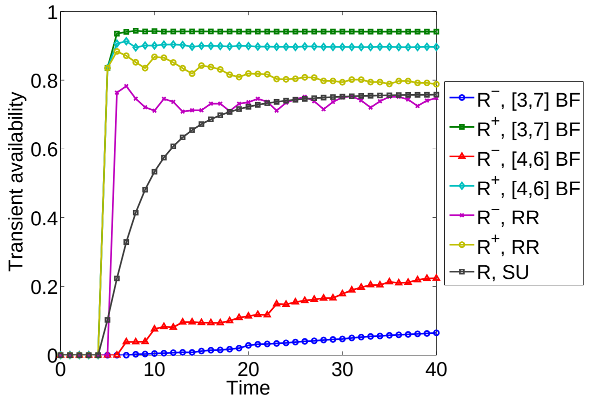

Figure 6: Transient availabilities for the leader election algorithm. Minimum and maximum availabilities are shown for two classes of bounded fair (BF) schedulers and the class of round-robin (RR) schedulers. The transient availability for the stationary uniform (SU) scheduler is also shown.

5.3 Tree-network leader election algorithm

Finally, we consider a leader election algorithm for tree-structured networks adapted from [20]. The algorithm establishes a directed tree in an anonymous network, by having each node select a "parent" node. This tree then has a unique root node (which selects itself as its parent), which is subsequently elected as leader. The selection of a *unique* node in such anonymous networks, however, depends on the choices made by the underlying scheduler. We embellish the algorithm with local faults that reset the "parent" of a node to a value chosen with a uniform distribution. Such a local fault occurs with a probability of 0.1 whenever a node executes a local step. We consider a network with five nodes. In the initial state each node has its "parent" variable set to itself.

Figure 6 depicts the probability that one node has been elected as leader. The availabilities are again similar to the previous case studies, although the impact of considering bounded fair schedulers is much greater here. This is most likely caused by the fact that this algorithm is only *weakly* self-stabilizing [20]. This means that there are schedulers for which the algorithm is not guaranteed to reach a safe configuration. In our setting, where we have transient faults and where we consider bounded fairness we can clearly see that there are bounded fair schedulers which greatly decrease the availability of the algorithm. We also see that the bounds of the transient availability drop in time for the RR scheduler class. This is due to the fact that with increasing time we have a higher probability of observing faults.

Figure 3 shows the transient availabilities for both the class of 1-RRRR and 2-RRRR schedulers. Similar observations as for the other cases can be made. However, we see here that the k-RRRR scheduler classes give much tighter bounds than the bounded fair scheduler classes. From our chosen starting configuration, the algorithm is always able to select a leader within one round, assuming no faults occur. This explains the much higher minimum transient availability for the k-RRRR scheduler classes for early time-points.

The long-run average availabilities for the leader election case study are listed in Table 7. As for the transient availabilites we see the large impact of bounded fairness on the long-run average availability bounds.

6 Related Work

6.1 Quantitative Analysis of Distributed Algorithms

Randomized distributed algorithms have been subjected to rigorous quantitative analysis ever since they were first reported in the literature. Qualitative properties of randomized distributed algorithms, such as *eventual termination* of an algorithm, are often less interesting than the

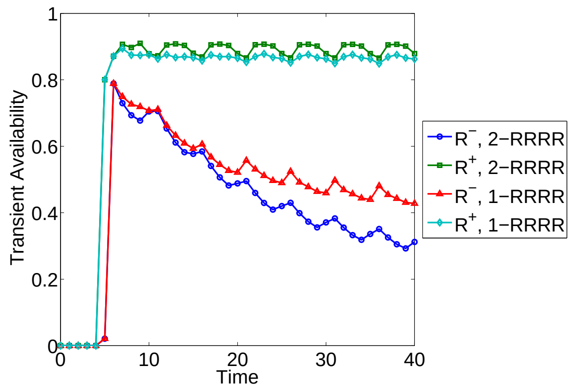

Figure 7: Transient availabilities for the leader-election algorithm. Minimum and maximum availabilities are shown for two classes of k-RRRR schedulers.

quantitative properties such as *probability* of reaching a good state in k steps or *expected number* of steps required to reach a good state. Gafni and Mitzenmacher [27] analyzed the effect of waiting time on the probability of a processor accessing its critical section for *timing-based mutual exclusion* algorithms in case the waiting times are governed by some random distribution. The expected number of rounds for a single token to circulate was calculated for a randomized token ring algorithm in [28]. It was also shown that the algorithm is *self-stabilizing* with *probability one*. A gossip-based shuffling algorithm is analyzed in [10] with respect to the number of nodes that receives data in each round. The subset of data to be sent and the subset of nodes to communicate within a round is selected randomly. Bakhshi and Fehnker also studied the impact of modeling assumptions by comparing results of PRISM with those derived by simulation and the analysis of ordinary differential equations in MATLAB. We refer reader to [42] for an elaborate survey on the analysis of distributed algorithms with probabilistic behavior.

There has recently been interest in the quantitative analysis of fault-tolerant distributed algorithms. Unlike randomized distributed algorithms discussed in the previous paragraph, stochastic behavior emanates from the (randomized) appearance of faults. Sorensen *et al.* [44] proposed methods to evaluate fault tolerance measures, such as reliability and expected time to a catastrophic fault, of systems specified as set of CSP processes. An algorithmic method is also described to derive an automaton from CSP-based specifications and subsequently transform the automaton to a Markov process. The method, however, addresses only systems with *permanent failures*, that is, a failure that cannot be corrected. A method to derive reliability and limiting availability of *self-stabilizing algorithms* with *intermittent transient faults* is presented in [22]. The method derives the equivalent DTMC by analyzing the self-stabilizing algorithm. This method has been further refined in [21] by automating the analysis with the help of PRISM. In addition to that, the authors also provide heuristics to re-engineer a self-stabilizing system if its fault tolerance measures below a certain threshold.

6.2 Communication Models

Correctness proofs of distributed protocols are drawn assuming either of two prevalent communication models: the message passing model and the shared memory model [39]. In the message passing model, two processes communicate with each other using a *channel* (also referred to as *link*) with the help of "`send`" and "`receive`" primitives. The shared memory model assumes that processes use shared memory variables that are exclusively meant for communication to exchange information. Among many variants of the shared memory model, the one that is used predominantly assumes that each process has two sets of communication registers: *read registers* and *write registers* [24]. Write registers are *owned* by a process and are used to inform neighbors about its local state. Read registers, on the other hand, are used to gather information about local states of neighboring

Case study	Schedulers	Min	Max
MST	[2,6] BF	0.553683	0.904349
	[3,5] BF	0.666325	0.820220
	2-RRRR	0.665750	0.832464
	1-RRRR	0.693737	0.799902
	RR	0.743096	0.754497
	SU	0.744122	0.744122
Gossip	[3,7] BF	0.663161	0.876990
	[4,6] BF	0.716577	0.820207
	2-RRRR	0.707631	0.833159
	1-RRRR	0.730030	0.807131
	RR	0.767610	0.769526
	SU	0.767789	0.767789
Leader	[3,7] BF	0.127691	0.891229
	[4,6] BF	0.336496	0.848096
	2-RRRR	0.298215	0.848437
	1-RRRR	0.447771	0.829705
	RR	0.763570	0.780348
	SU	0.760131	0.760131

Table 7: Extremal long-run average availabilities for the three case studies for various bounded fair (BF), round-robin (RR), and stationary uniform (SU) classes of schedulers. Note that for the stationary uniform scheduler we find an exact value, since it is a single scheduler rather than a scheduler class.

processes.

The communication model used by *population protocols* [5] is slightly different from the classical models described above. Population protocols are meant to run on *uniform* and *anonymous* distributed systems with a large number of mobile processes that have limited resources and no control over their movement. The underlying communication model simply assumes that processes communicate *simultaneously* using an *interaction* primitive. This abstraction leaves the choice of implementing communication primitive via shared memory or message passing open.

6.3 Failure Models

Fault-tolerant distributed algorithms are typically designed using a fault model that assumes that at most t out of n components can fail at system runtime [39, 17]. Furthermore, the model assumes that processes and links fail independently of each other. A process can fail either by simply crashing or by acting in an arbitrary manner (Byzantine failure) [38]. Another underlying assumption is that the likelihood of a component failure is independent of the system runtime. A variant of the classical fault model is the "Heard-Of Model," which nullifies the distinction between a "benign" process failure and a link failure and subsumes all non-arbitrary failures under transmission failures [12].

The failure model remains essentially the same even for distributed protocols which provide probabilistic dependability—such as gossip protocols [18]—instead of tight fault-masking guaranties. The underlying fault model of gossip protocols assumes that the ratio of failed to non-failed processes cannot be greater than certain threshold $q(0 < q < 1)$ [25]. It is also assumed that message loss failures are independent of each other [3].

The models presented above are geared towards easing the task of drawing correctness proofs of the properties of distributed protocols. However, these models do not always capture realistic scenarios well enough [33, 3]. This claim has also been strengthened by the empirical studies which have shown that process failures are not necessarily independent of each other [47].

In order to provide abstractions closer to reality, modified fault models have been presented which assume different failure semantics for the various system components. The *hybrid fault model* assumes that at most b processes can exhibit a crash failure and at most m processes out of n processes behave arbitrarily [40]. There are also failure models that capture *undirected dependencies* [32] and *directed dependencies* [46] between the failures of system components.

The restriction on the number of component failures is somewhat relaxed when proving that a protocol is self-stabilizing. A self-stabilizing protocol converges under an *arbitrary* number of *transient* faults [24]. Transient faults are temporary and only modify the state of the processes. This assumption holds for self-stabilizing versions of population protocols as well [6, 11].

6.4 Fairness Notions

The concept of fairness has been studied extensively as it facilitates discarding certain unrealistic execution paths while verifying liveness properties of distributed algorithms. To that end, various notions of fairness have been proposed in literature. In a broad sense, fairness ensures that an action or a process is activated sufficiently often if it has been enabled often enough [26]. *Weak fairness* ensures that a *continuously* enabled transition is taken *infinitely* often. An *infinitely* often enabled transition is activated *infinitely* often under *strong fairness*. It has been observed that in many instances fairness alone is not sufficient to guarantee that a transition is eventually activated because it is not enabled long enough due to "race conditions" [7]. The notion of *hyperfairness* is proposed to exclude such pathological traces where a transition is not enabled due to intermittent unavailability of the required resources. Hyperfairness is deployed with an underlying notion of fairness. It guarantees that a transition is enabled sufficiently enough so that it can be activated by the underlying fairness notion. This notion of hyperfairness is generalized in [37]. Lamport's notion of hyperfairness ensures that a transition is activated *infinitely* often if it is *infinitely often possible*. The relative strengths of various notions of fairness have also been studied [45]. The notion of *probabilistic fairness* has been defined for a class of distributed algorithms [5, 13]. A *probabilistic* scheduler resolves non-determinism by choosing a next transition according to certain probability distribution. It has been shown that a probabilistic scheduler is *fair with probability* 1 if each transition has non-zero probability of being selected *and* the probability distribution remains unchanged during the system runtime [13].

All the above notions of fairness only require certain actions to take place *at some time-point* under various conditions, but generally do not enforce these actions to occur in a particular time-frame. *Bounded fairness*, on the other hand, guarantees that a *continuously* enabled transition is activated within a *bounded* number of steps and is stronger than the notion of weak fairness [2, 29, 19]. The notion of bounded fairness has been used to prove various qualitative properties of distributed algorithms such as consensus in asynchronous systems [9, 1, 8].

7 Conclusion

This paper has discussed the notion of bounded fairness for probabilistic models of distributed algorithms. Bounded fairness naturally captures clock drift in near-symmetric distributed systems. We have shown how to apply these new fairness notions to models of distributed algorithms and discussed properties of our scheduling classes. We have developed the theory of bounded fairness for Markov decision processes as well as the practice of applying this scheduler class to models in a guarded command language.

In several case studies we have shown that customary ways of fixing an execution order, such as round-robin scheduling and random scheduling, may lead to too optimistic or too pessimistic estimations of quantitative properties of distributed algorithms. On the other hand, we see that using bounded fairness increases the size of our models and makes analysis of algorithms operating on large networks difficult. For this reason, it is useful to search for efficient approximations of quantitative properties for the class of bounded fair schedulers. Another interesting consequence of bounded fairness for MDPs is that it now allows us to study the quantitative effect of clock drift on specific distributed algorithms. One avenue to investigate is whether the resilience of different distributed algorithms to clock drift can be quantified. Another extension of this work can be the analysis of the distributed algorithms which exhibit certain qualitative properties only under a specific subset of schedulers and the study of bounded fair schedulers which maximize or minimize the dependability metrics related to such qualititative properties.

References

[1] R. Alur, H. Attiya, and G. Taubenfeld. Time-Adaptive Algorithms for Synchronization. *SIAM J. Comput.*, 26(2):539–556, 1997.

[2] R. Alur and T. A. Henzinger. Finitary Fairness. *ACM Trans. Programming Languages and Systems*, 20(6):1171–1194, 1998.

[3] L. Alvisi, J. Doumen, R. Guerraoui, B. Koldehofe, H. Li, R. van Renesse, and G. Tredan. How robust are gossip-based communication protocols? *Operating Systems Review*, 41(5):14–18, 2007.

[4] T. R. Andel and A. Yasinsac. On the credibility of manet simulations. *IEEE Computer*, 39(7):48–54, 2006.

[5] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[6] D. Angluin, J. Aspnes, M. J. Fischer, and H. Jiang. Self-stabilizing population protocols. In *Int'l Symp. on Principles of Distributed Systems*, volume 3974 of *LNCS*, pages 103–117. Springer, 2005.

[7] P. C. Attie, N. Francez, and O. Grumberg. Fairness and hyperfairness in multi-party interactions. *Distributed Computing*, 6(4):245–254, 1993.

[8] H. Attiya and T. Djerassi-Shintel. Time Bounds for Decision Problems in the Presence of Timing Uncertainty and Failures. *J. Parallel Distrib. Comput.*, 61(8):1096–1109, 2001.

[9] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. In *23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 359–369. ACM Press, 1991.

[10] R. Bakhshi and A. Fehnker. On the impact of modelling choices for distributed information spread. a comparative study. In *6th Int'l Conf. on Quantitative Evaluation of Systems (QEST)*, pages 41–50. IEEE CS, 2009.

[11] J. Beauquier, J. Burman, and S. Kutten. Making population protocols self-stabilizing. In *SSS*, volume 5873 of *LNCS*, pages 90–104. Springer, 2009.

[12] B. Charron-Bost and A. Schiper. The heard-Of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

[13] I. Chatzigiannakis, S. Dolev, S. P. Fekete, O. Michail, and P. G. Spirakis. Not all fair probabilistic schedulers are equivalent. In *Int'l Conf. on Principles of Distributed Systems*, LNCS. Springer, 2009. (accepted).

[14] A. Coccoli, P. Urbán, and A. Bondavalli. Performance analysis of a consensus algorithm combining stochastic activity networks and measurements. In *DSN*, pages 551–560. IEEE CS, 2002.

[15] U. M. Colesanti, C. Crociani, and A. Vitaletti. On the accuracy of omnet++ in the wireless sensornetworks domain: simulation vs. testbed. In *PE-WASUN '07*, pages 25–31, 2007.

[16] Z. Collin and S. Dolev. Self-stabilizing depth-first search. *IPL*, 49(6):297–301, 1994.

[17] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. When birds die: Making population protocols fault-tolerant. In *DCOSS*, volume 4026 of *LNCS*, pages 51–66. Springer, 2006.

[18] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC*, pages 1–12. ACM Press, 1987.

[19] N. Dershowitz, D. N. Jayasimha, and S. Park. Bounded fairness. In *Verification: Theory and Practice*, volume 2772 of *LNCS*, pages 304–317. Springer, 2003.

[20] S. Devismes, S. Tixeuil, and M. Yamashita. Weak vs. self vs. probabilistic stabilization. In *28th Int'l Conf. on Distributed Computing Systems (ICDCS)*, pages 681–688, Washington, DC, USA, 2008. IEEE CS.

[21] A. Dhama, O. Theel, P. Crouzen, H. Hermanns, R. Wimmer, and B. Becker. Dependability engineering of silent self-stabilizing systems. In *SSS*, volume 5873 of *LNCS*, pages 238–253. Springer, 2009.

[22] A. Dhama, O. Theel, and T. Warns. Reliability and availability analysis of self-stabilizing systems. In *SSS*, volume 4280 of *LNCS*, pages 244–261. Springer, 2006.

[23] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Commun. ACM*, 8(9):569, 1965.

[24] S. Dolev. *Self-Stabilization*. The MIT Press, 2000.

[25] X. Fan, J. Cao, W. Wu, and M. Raynal. On modeling fault tolerance of gossip-based reliable multicast protocols. In *ICPP*, pages 149–156. IEEE CS, 2008.

[26] N. Francez. *Fairness*. Springer, 1986.

[27] E. Gafni and M. Mitzenmacher. Analysis of timing-based mutual exclusion with random times. *SIAM J. Comp.*, 31(3):816–837, 2001.

[28] T. Herman. Probabilistic Self-Stabilization. *Inf. Process. Lett.*, 35(2):63–67, 1990.

[29] W. H. Hesselink. Progress under bounded fairness. *Distributed Computing*, 12(4):197–207, 1999.

[30] A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *TACAS*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[31] P. Hurni and T. Braun. Calibrating wireless sensor network simulation models with real-world experiments. In *Proc. of the 8th Int'l IFIP-TC 6 Networking Conf.*, pages 1–13. Springer, 2009.

[32] F. P. Junqueira and K. Marzullo. Designing algorithms for dependent process failures. In *Future Directions in Distr. Comp., Research and Position Papers*, volume 2584 of *LNCS*, pages 24–28. Springer, 2003.

[33] I. Keidar and K. Marzullo. The need for realistic failure models in protocol design. In *Information Survivability Workshop*, 2002.

[34] A. Köpke, M. Swigulski, K.Wessel, D.Willkomm, P. Haneveld, T. Parker, O. Visser, H. Lichte, and S. Valentin. Simulating wireless and mobile networks in omnet++: The mixim vision. In *1st Int'l Workshop on OMNeT++*, 2008.

[35] S. Kurkowski, T. Camp, and M. Colagrosso. Manet simulation studies: the incredibles. *MC2R*, 9(4):50–61, 2005.

[36] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *STTT*, 6(2):128–142, 2004.

[37] L. Lamport. Fairness and hyperfairness. *Distributed Computing*, 13(4):239–245, 2000.

[38] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine generals problem. *TOPLAS*, 4(3):382–401, 1982.

[39] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[40] F. J. Meyer and D. K. Pradhan. Consensus with dual failure modes. *TPDS*, 2(2):214–222, 1991.

[41] G. E. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[42] G. Norman. Analysing Randomized Distributed Algorithms. In *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 384–418. Springer, 2004.

[43] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[44] E. V. Sorensen, J. Nordahl, and N. H. Hansen. From CSP to Markov models. *IEEE. Trans. Software Eng.*, 19(6):554–570, 1993.

[45] H. Völzer. Refinement-robust fairness. In *Int'l Conf. on Concurrency Theory*, volume 2421 of *LNCS*, pages 547–561. Springer, 2002.

[46] T. Warns, F. C. Freiling, and W. Hasselbring. Solving consensus using structured failure models. In *SRDS*, pages 212–224. IEEE CS, 2006.

[47] T. Warns, C. Storm, and W. Hasselbring. Availability of globally distributed nodes: An empirical evaluation. In *SRDS*, pages 279–284. IEEE CS, 2008.

[48] R. Wimmer, B. Braitling, B. Becker, E. M. Hahn, P. Crouzen, H. Hermanns, A. Dhama, and O. Theel. Symblicit calculation of long-run averages for concurrent probabilistic systems. In *7th Int'l Conf. on Quantitative Evaluation of Systems (QEST)*, 2010. Submitted.

A Proof of Theorem 1

Given a MDP $M = (S, A, R)$, a function F from paths of M to sets of actions, and the scheduler class C_F characterized by F we construct the MDP $M' = (S', A, R')$ such that the extremal probabilities of M over C_F match the extremal probabilities of M' over all schedulers.

Let the state space of M' be the set of finite paths of M: $S' = (S \times A)^* \times S$. A path ρ of M' is then in a sense a path of paths, i.e. $\rho \in (((S \times A)^* \times S) \times A)^* \times ((S \times A)^* \times S)$. We say that a path $\rho = \sigma_1 a_1 \sigma_2 a_2 \ldots \sigma_n$ is *coherent* if it starts with a path σ_1 of M of length 0 (i.e. a single state) and for each subsequent path σ_i of M in ρ we have that its predecessor σ_{i-1} in ρ is also its immediate prefix and the last action of σ_i is the action between σ_{i-1} and σ_i in ρ. As an example a path ρ of M' with $\rho = s_1, a_1, s_1 a_1 s_2, a_2, s_1 a_1 s_2 a_2 s_3$ (here the constituents of ρ are separated by commas) is coherent. It is clear that if a path of M' is coherent it is competely determined by its last state. For a state $\sigma \in S'$ we write $\gamma(\sigma)$ for the coherent path such that $\mathit{last}(\gamma(\sigma)) = \sigma$.

The transition relation R' follows the transition relation of R, lifted to paths, but it excludes those transitions which violate the restrictions of the function F.

$$R' = \{(\sigma, a, \pi') \mid \sigma \in ((S \times A)^* \times S) \ \wedge (\mathit{last}(\sigma), a, \pi) \in R \wedge a \in F(\sigma)\},$$

Where π' is defined to match π, lifted to coherent paths, such that for all states $s \in S$ we have:

$$P(\bar{R}'(\sigma, a) = \sigma a s) = P(\bar{R}(\mathit{last}(\sigma), a) = s).$$

Note that for each state $\sigma' = \sigma a s$ in S' there is exactly one state which has a transition to σ', namely the state σ. States of M' which represent paths of length 0 of M have no such incoming transitions.

We now prove two lemmas which relate schedulers of M' to schedulers of M in C_F by considering the transition probabilities of their induced Markov chains.

Lemma 1. *Given a scheduler f for M such that $f \in C_F$. Let the function f' from paths of M' to distributions over the actions A be such that for any path σ of M and any action $a \in A$ we have:*

$$P\big(\bar{f}'(\gamma(\sigma)) = a\big) = P(\bar{f}(\sigma) = a\big).$$

For non-coherent paths ρ the function f' may choose any distribution over the set of actions enabled in the last state of ρ. Now we have that any such f' is a scheduler of M' and for any path σ of M, action $a \in A$, state $s \in S$, and time-point $k \in \mathbb{Z}_{\geq 0}$ we have that:

$$P\big(X^{(k+1)}_{f',M'} = \gamma(\sigma a s) \mid X^{(k)}_{f',M'} = \gamma(\sigma)\big) = P\big(X^{(k+1)}_{f,M} = \sigma a s \mid X^{(k)}_{f,M} = \sigma\big)$$

Proof. We first show that f' is a scheduler of M'. It is then required for any path ρ of M' that f' assigns a probability greater than zero only to those actions which are enabled in the last state of ρ. For non-coherent paths this holds by definition. For a coherent path ρ we have that there exists some path σ of M such that $\rho = \gamma(\sigma)$. From the definition of R' we know that the actions enabled for σ in M' are exactly the actions in $F(\sigma)$. For an action not in $F(\sigma)$ we have that: $P(\bar{f}'(\gamma(\sigma)) = a) = P(\bar{f}(\sigma) = a) = 0$, by the fact that f is in C_F. This shows that f' is a scheduler of M'.

For the transition probabilities of the MC induced by the scheduler f' we find the following. For any path σ of M, any action a in $F(\sigma)$, state $s \in S$, and time-point $k \in \mathbb{Z}_{\geq 0}$ we have, by applying the definitions of f' and R', that:

$$\begin{aligned} P(X^{(k+1)}_{f',M'} = \gamma(\sigma a s) \mid X^{(k)}_{f',M'} = \gamma(\sigma)) &= P(\bar{f}'(\gamma(\sigma)) = a) \cdot P(\bar{R}'(\sigma, a) = \sigma a s) \\ &= P(\bar{f}(\sigma) = a) \cdot P(\bar{R}(\mathit{last}(\sigma), a) = s) \\ &= P(X^{(k+1)}_{f,M} = \sigma a s \mid X^{(k)}_{f,M} = \sigma) \end{aligned}$$

For an action $a \notin F(\sigma)$ we have that $P(\bar{R}'(\sigma, a) = \sigma a s)$ and $P(\bar{f}(\sigma) = a)$ are both zero and thus the transition probabilities for both $X_{f',M'}$ and $X_{f,M}$ are zero. □

Lemma 2. *Given a scheduler f' for M'. Let the function f from paths of M to distributions over A be such that for any path σ of M and any action $a \in A$ we have:*

$$P(\bar{f}(\sigma) = a) = P(\bar{f}'(\gamma(\sigma)) = a).$$

Now we have that any such f is a scheduler of M and a member of C_F. Furthermore, for any path σ of M, action $a \in A$, state $s \in S$, and time-point $k \in \mathbb{Z}_{\geq 0}$ we have that:

$$P(X_{f,M}^{(k+1)} = \sigma as \mid X_{f,M}^{(k)} = \sigma) = P(X_{f',M'}^{(k+1)} = \gamma(\sigma as) \mid X_{f',M'}^{(k)} = \gamma(\sigma))$$

Proof. It is trivial that f is a scheduler of M since it assigns a distribution over actions to every path of M. As for Lemma 1 we have that scheduler f' can only assign non-zero probabilities to actions enabled for a path $\gamma(\sigma)$ which is exactly the set $F(\sigma)$. It follows that f is a member of C_F. The proof that the transition probabilities for $X_{f,M}$ match those for $X_{f',M'}$ is identical to the one for Lemma 1. □

Now we prove two lemmas which relate the transient probabilities induced from M with the transient probabilities induced from M'.

Lemma 3. *Given a scheduler f for M such that $f \in C_F$, a starting state $s \in S$ and a set of goal paths $\mathcal{G} \subset (S \times A)^* \times S$. Let the scheduler f' be the scheduler derived from f as in Lemma 1. We now have for any time-point $k \in \mathbb{Z}_{\geq 0}$ the following:*

$$P(X_{f,M}^{(k)} \in \mathcal{G} \mid X_{f,M}^{(0)} = s) = P(X_{f',M'}^{(k)} \in \gamma(\mathcal{G}) \mid X_{f',M'}^{(0)} = \gamma(s)).$$

Note that the state s is used here as a path of length 0 and we use the notation $\gamma(\mathcal{G}) = \{\gamma(\sigma) \mid \sigma \in \mathcal{G}\}$.

Proof. We prove Lemma 3 by induction on the time-point k. As the base case we have $k = 0$ for which we find: $P(X_{f,M}^{(0)} \in \mathcal{G} \mid X_{f,M}^{(0)} = s)$ equals one if $s \in \mathcal{G}$ and zero otherwise. For the induced Markov chain $X_{f',M'}$ we find: $P(X_{f',M'}^{(0)} \in \gamma(\mathcal{G}) \mid X_{f',M'}^{(0)} = \gamma(s))$ equals one if $\gamma(s) \in \gamma(\mathcal{G})$ and zero otherwise. Since s is a path of length zero we have $\gamma(s) = s$ and $s \in \gamma(\mathcal{G})$ if and only if $s \in \mathcal{G}$. Thus we have that $P(X_{f,M}^{(0)} \in G \mid X_{f,M}^{(0)} = s)$ equals $P(X_{f',M'}^{(0)} \in g(G) \mid X_{f',M'}^{(0)} = h(s))$ for all states $s \in S$ and all sets of paths $\mathcal{G}$.

Given a fixed arbitrary time-point $k > 0$ we use the following induction assumption, that for all states $s \in S$ and all sets of paths $\mathcal{H} \subset (S \times A)^* \times S$ we have:

$$P(X_{f,M}^{(k-1)} \in \mathcal{H} \mid X_{f,M}^{(0)} = s) = P(X_{f',M'}^{(k-1)} \in \gamma(\mathcal{H}) \mid X_{f',M'}^{(0)} = \gamma(s)).$$

Our goal is now to prove that, given that the induction assumption holds, the following can be proven for all states s and all sets of paths $\mathcal{G}$:

$$P(X_{f,M}^{(k)} \in \mathcal{G} \mid X_{f,M}^{(0)} = s) = P(X_{f',M'}^{(k)} \in \gamma(\mathcal{G}) \mid X_{f',M'}^{(0)} = \gamma(s)).$$

For the probabilities at time-point k we have:

$$\begin{aligned} &P(X_{f,M}^{(k)} \in \mathcal{G} \mid X_{f,M}^{(0)} = s) \\ &\quad = \sum_{\sigma' \in (S \times A)^{k-1} \times S} P(X_{f,M}^{(k)} \in \mathcal{G} \mid X_{f,M}^{(k-1)} = \sigma') \cdot P(X_{f,M}^{(k-1)} = \sigma' \mid X_{f,M}^{(0)} = s) \end{aligned}$$

And:

$$\begin{aligned} &P(X_{f',M'}^{(k)} \in \gamma(\mathcal{G}) \mid X_{f',M'}^{(0)} = \gamma(s)) \\ &\quad = \sum_{\rho' \in (S' \times A)^{k-1} \times S'} P(X_{f',M'}^{(k)} \in \gamma(\mathcal{G}) \mid X_{f',M'}^{(k-1)} = \rho') \cdot P(X_{f',M'}^{(k-1)} = \rho' \mid X_{f',M'}^{(0)} = \gamma(s)) \end{aligned}$$

All the paths in $\gamma(\mathcal{G})$ are coherent and therefor the probability $P(X^{(k)}_{f',M'} \in \gamma(\mathcal{G}) \mid X^{(k-1)}_{f',M'} = \rho')$ is zero for non-coherent paths ρ'. We then have that we can restrict the summation to just the coherent paths:

$$\begin{aligned} &P(X^{(k)}_{f',M'} \in \gamma(\mathcal{G}) \mid X^{(0)}_{f',M'} = \gamma(s)) \\ &\quad = \sum_{\sigma' \in (S\times A)^{k-1}\times S} P(X^{(k)}_{f',M'} \in \gamma(\mathcal{G}) \mid X^{(k-1)}_{f',M'} = \gamma(\sigma')) \cdot P(X^{(k-1)}_{f',M'} = \gamma(\sigma') \mid X^{(0)}_{f',M'} = \gamma(s)) \end{aligned}$$

Since f is in C_F we can apply Lemma 1 and the induction assumption to find:

$$\begin{aligned} &P(X^{(k)}_{f',M'} \in \gamma(\mathcal{G}) \mid X^{(0)}_{f',M'} = \gamma(s)) \\ &\quad = \sum_{\sigma' \in (S\times A)^{k-1}\times S} P(X^{(k)}_{f',M'} \in \gamma(\mathcal{G}) \mid X^{(k-1)}_{f',M'} = \gamma(\sigma')) \cdot P(X^{(k-1)}_{f',M'} = \gamma(\sigma') \mid X^{(0)}_{f',M'} = \gamma(s)) \\ &\quad = \sum_{\sigma' \in (S\times A)^{k-1}\times S} P(X^{(k)}_{f,M} \in \mathcal{G} \mid X^{(k-1)}_{f,M} = \sigma') \cdot P(X^{(k-1)}_{f,M} = \sigma' \mid X^{(0)}_{f,M} = s) \\ &\quad = P(X^{(k)}_{f,M} \in \mathcal{G} \mid X^{(0)}_{f,M} = s) \end{aligned}$$

Note that we use the fact that any path σ'' in $\mathcal{G}$ where $P(X^{(k)}_{f,M} = \sigma'' \mid X^{(k-1)}_{f,M} = \sigma')$ is non-zero must be a one step extension of σ', i.e. of the form $\sigma'' = \sigma' as$. This enables us to apply Lemma 1. This proves the induction step and thus Lemma 3. □

Lemma 4. *Given a scheduler f' for M', a starting state $s \in S$ and a set of goal paths $\mathcal{G} \subset (S \times A)^* \times S$. Let the scheduler f be the scheduler derived from f' as in Lemma 2. We now have for any time-point $k \in \mathbb{Z}_{\geq 0}$ the following:*

$$P(X^{(k)}_{f',M'} \in \gamma(G) \mid X^{(0)}_{f',M'} = \gamma(s)) = P(X^{(k)}_{f,M} \in \mathcal{G} \mid X^{(0)}_{f,M} = s).$$

We again use the state s as path of length zero and use the notation $\gamma(G) = \{\gamma(\sigma) \mid \sigma \in \mathcal{G}\}$.

The proof for Lemma 4 is identical to the proof for Lemma 3 except that Lemma 2 is used instead of Lemma 1 to prove the induction step.

Lemmas 3 and 4 tell us that for every scheduler in C_F for M we find an identical, with respect to transient probabilities, scheduler for M' and vice versa. Here, states in the induced MC for M are paths in M, while states in the induced MC of M' are "paths of paths" of M, but since we restrict to coherent "paths of paths", which are uniquely defined by their last entry, there is a one-to-one correspondence between the two induced MCs. It then immediately follows that the set of all transient probabilities for M and M' are equal and then their infimum and supremum must be equal as well. By a similar reasoning we have that the infimum and supremum long-run average probabilities must be the same which then proves Theorem 1.

B Proof of Theorem 2

Given an enabled MDP $M = (S, A, R)$ with $|A| = N$ and given bounds $L, U \in \mathbb{Z}$ with $1 \leq L \leq N$ and $N \leq U$, we prove that a scheduler f is $[L, U]$ bounded fair if and only if for all time-points $k \in \mathbb{Z}_{\geq 0}$ the period of the induced Markov chain at time-point k lies between L and U:

$$P\left(\Lambda^{(k)}_{f,M} \in [L, U]\right) = 1,$$

And for any action a the first occurrence of a is less than or equal to U:

$$P\left(\Delta^{(a)}_{f,M} \leq U\right) = 1.$$

Before we prove Theorem 2 we first introduce the following lemma.

Lemma 5. *Given an enabled MDP $M = (S, A, R)$ with $|A| = N$ and given bounds $L, U \in \mathbb{Z}$ with $1 \leq L \leq N$ and $N \leq U$. For any $[L, U]$ bounded fair scheduler f we have that for any finite path $\sigma \in (S \times A)^* \times S$ of length k the following holds. If the length of σ is larger or equal to $U - 1$, then the number of actions that did not occur in the last $U - 1$ steps is less than or equal to one. If the length of σ is smaller than $U - 1$ then the number of actions that did not occur in σ is less than or equal to $U - k$. This means that the probability of reaching a path which violates these restrictions is zero:*

$$k \geq U - 1 \land |A \setminus A_\sigma^{(U-1)}| > 1 \Rightarrow P(X_{f,M}^{(k)} = \sigma) = 0$$
$$k < U - 1 \land |A \setminus A_\sigma^{(k)}| > U - k \Rightarrow P(X_{f,M}^{(k)} = \sigma) = 0$$

Proof. We prove Lemma 5 by induction on the path length k. For the base case $k = 0$ we have that $A_\sigma^{(0)} = \emptyset$ and then $|A \setminus A_\sigma^{(k)}| = N \leq U$ which means the lemma holds for the case $k = 0$.

Now we fix an arbitrary path length $k > 0$ and use as our induction assumption that for $k - 1$ we have that the lemma holds. We now prove that the lemma holds for an arbitrary path σ' of length k. Since k is greater than zero, we have that $\sigma' = \sigma as$, where σ is a path of M of lenght $k - 1$, a is some action in A and s is a state in S. We now consider two possibilities for the value of k:

- **Case $k < U$.** Here we have that $k-1 < U-1$ and then we have by the induction assumption that for the path σ of length $k-1$ the following holds: $|A \setminus A_\sigma^{(k-1)}| > U-k+1 \Rightarrow P(X_{f,M}^{(k-1)} = \sigma) = 0)$. Now the set of actions in the last k steps of σ' is equal to the set of actions in the last $k - 1$ steps of σ including a: $A_{\sigma'}^{(k)} = A_\sigma^{(k-1)} \cup \{a\}$. We now consider whether a was already in the set $A_\sigma^{(k-1)}$ or not.
  - For the case $a \notin A_\sigma^{(k-1)}$ we have that $|A \setminus A_{\sigma'}^{(k)}| = |A \setminus A_\sigma^{(k-1)}| - 1$. Now if we assume that $|A \setminus A_{\sigma'}^{(k)}| > U - k$ then we have that $|A \setminus A_\sigma^{(k-1)}| > U - k + 1$, which gives us, by the induction assumption, that: $P(X_{f,M}^{(k-1)} = \sigma) = 0)$. Since σ' can only be reached via σ we have $P(X_{f,M}^{(k)} = \sigma') = 0)$ which proves the lemma for this case.
  - For the case $a \in A_\sigma^{(k-1)}$ we have that $|A \setminus A_{\sigma'}^{(k)}| = |A \setminus A_\sigma^{(k-1)}|$. We now consider three possibilities for the size of the set $A \setminus A_\sigma^{(k-1)}$:
    - For $|A \setminus A_\sigma^{(k-1)}| > U - k + 1$ we have that the probability to reach path σ is zero and then the probability to reach path σ' is also zero.
    - For $|A \setminus A_\sigma^{(k-1)}| < U - k + 1$ we have that $|A \setminus A_{\sigma'}^{(k)}| \leq U - k$. We now consider the size of k and show that in all cases the left-hand sides of the implications are falsified:
      - For $k < U - 1$ we must consider the second implication and find that $|A \setminus A_{\sigma'}^{(k)}| \leq U - k$ immediately falsifies it.
      - For $k = U - 1$ we must consider the first implication and we have that $U - k = 1$. Now we have that $|A \setminus A_{\sigma'}^{(k)}| = |A \setminus A_{\sigma'}^{(U-1)}| \leq 1$ falsifying the left-hand side of the first implication.
    - For the case $|A \setminus A_\sigma^{(k-1)}| = U - k + 1$ we have that $F(\sigma') = A \setminus A_\sigma^{(k-1)}$ but then action a, which we now consider to be in $A_\sigma^{(k-1)}$, is selected by scheduler f with probability zero. We then have $P(X_{f,M}^{(k)} = \sigma as) = P(\bar{f}(\sigma) = a) \cdot P(\bar{R}(\sigma, a) = s) \cdot P(X_{f,M}^{(k-1)} = \sigma) = 0$ since $P(\bar{f}(\sigma) = a) = 0$.

  This shows that the induction step holds for paths shorter than U.

- Consider now the case $k \geq U$ then we have that $k - 1 \geq U - 1$ and by the induction assumption we have that the following holds: $|A \setminus A_\sigma^{(U-1)}| > 1 \Rightarrow P(X_{f,M}^{(k-1)} = \sigma) = 0)$. The set of actions in the last $U - 1$ steps of $\sigma' = \sigma as$ is equal to the set of actions in the last

$U-2$ steps of σ with the addition of action a: $A_{\sigma'}^{(U-1)} = A_{\sigma}^{(U-2)} \cup \{a\}$. For the set $A_{\sigma}^{(U-2)}$ we have that $|A_{\sigma}^{(U-1)}| - 1 \leq |A_{\sigma}^{(U-2)}| \leq |A_{\sigma}^{(U-1)}|$. We now consider three possibilities for the size of the set $A \setminus A_{\sigma}^{(U-2)}$ and its inclusion of the action a:

- For the case that $|A \setminus A_{\sigma}^{(U-2)}| > 2$ we have that $|A \setminus A_{\sigma}^{(U-1)}| > 1$ and then $P(X_{f,M}^{(k-1)} = \sigma) = 0$ by the induction assumption and then we also have that $P(X_{f,M}^{(k)} = \sigma as) = 0$ proving that the lemma holds.
- For the case $|A \setminus A_{\sigma}^{(U-2)}| < 2$ or the case $|A \setminus A_{\sigma}^{(U-2)}| = 2$ and $a \notin A_{\sigma}^{(U-2)}$ we have that $|A \setminus A_{\sigma'}^{(U-1)}| < 2$ and thus the lemma holds.
- Finally we consider the case where $|A \setminus A_{\sigma}^{(U-2)}| = 2$ and $a \in A_{\sigma}^{(U-2)}$. It is trivial that we also have $a \in A_{\sigma}^{(U-1)}$. For the set $|A \setminus A_{\sigma}^{(U-1)}|$ we again have two possibilities.
  - In case $|A \setminus A_{\sigma}^{(U-1)}| = 2$, the probability to reach path σ, and thus σ', is again zero.
  - For the case $|A \setminus A_{\sigma}^{(U-1)}| = 1$ we have that $F(\sigma) = A \setminus A_{\sigma}^{(U-1)}$ but then $a \notin F(\sigma)$ and we again find that $P(X_{f,M}^{(k)} = \sigma as)$ is zero, because scheduler f selects a after path σ with probability zero.

This proves that the induction step holds and thus Lemma 5 holds. □

We are now ready to prove Theorem 2.

Proof. We prove the two implications separately.

Bounded fair schedulers induce bounded periods We first show that if a scheduler is $[L, U]$ bounded fair then its period always lies between L and U with probability one. To do this we prove that for any k the probability that the period of the induced MC $X_{f,M}$ is smaller than L or greater than U is zero and that the probability that the first occurence of an action is greater than U is zero.

- For the probability of the period being smaller than L we have that:

$$P(\Lambda_{f,M}^{(k)} < L) = \sum_{a \in A} P(\bar{f}(X_{f,M}^{(k)}) = a \ \wedge (\bar{f}(X_{f,M}^{(k+1)}) = a \vee \ldots \vee \bar{f}(X_{f,M}^{(k+L-1)}) = a))$$
$$\leq \sum_{i=1}^{L-1} \sum_{a \in A} P(\bar{f}(X_{f,M}^{(k)}) = a \wedge \bar{f}(X_{f,M}^{(k+i)}) = a)$$

Consider a path $\sigma \in (S \times A)^k \times S$ of length k, an arbitrary fixed action $a \in A$ and the path $\sigma' = \sigma a \sigma''$ where $\sigma'' \in (S \times A)^{i-1} \times S$ has length $i-1$ with $1 \leq i < L$. Note that path σ' has length $k+i$, that a is in $A_{\sigma'}^{i}$ and that all paths of length greater than zero can be described in this way.

We now look at the possibilities for the allowed actions for path σ', $F(\sigma')$, by going over the three cases in Definition 7. We then show that in all cases $a \notin F(\sigma')$.

1. If $k+i \geq U \wedge |A \setminus A_{\sigma'}^{(U-1)}| = 1$ then we have $F(\sigma') = A \setminus A_{\sigma'}^{(U-1)}$. Since $i < L \leq U$ we have that $a \in A_{\sigma'}^{(U-1)}$ and then $a \notin F(\sigma')$.
2. If $k+i < L$ or $k+i < U \wedge |A \setminus A_{\sigma'}^{(k+i)}| = U - (k+i)$ then $F(\sigma') = A \setminus A_{\sigma'}^{(k+i)}$. Obviously we have that $a \in A_{\sigma'}^{(k+i)}$ and then $a \notin F(\sigma')$.
3. If cases 1 and 2 don't apply we have $A \setminus A_{\sigma'}^{(L-1)}$. Since $i < L$ we have $a \in A_{\sigma'}^{(L-1)}$ and again $a \notin F(\sigma')$.

We have now proven that $a \notin F(\sigma')$. For scheduler f this proves that $P(\bar{f}(\sigma') = a) = 0$. We now have:

$$\begin{aligned} &P(X^{(k)}_{f,M} = \sigma \wedge X^{(k+i)}_{f,M} = \sigma a \sigma'' \wedge \bar{f}(X^{(k)}_{f,M}) = a \wedge \bar{f}(X^{(k+i)}_{f,M}) = a) \\ &\quad = P(X^{(k)}_{f,M} = \sigma \wedge X^{(k+i)}_{f,M} = \sigma a \sigma'' \wedge \bar{f}(\sigma) = a \wedge \bar{f}(\sigma a \sigma'') = a) = 0 \end{aligned}$$

For the probability of repeating an action within L steps we now find:

$$\begin{aligned} &\sum_{i=1}^{L-1} \sum_{a \in A} P(f(X^{(k)}_{f,M}) = a \wedge \bar{f}(X^{(k+i)}_{f,M}) = a) \\ &\quad = \sum_{i=1}^{L-1} \sum_{a \in A} \sum_{\sigma \in Paths^{(k)}_M} \sum_{\sigma'' \in Paths^{(i-1)}_M} P(X^{(k)}_{f,M} = \sigma \\ &\qquad \wedge X^{(k+i)}_{f,M} = \sigma a \sigma'' \wedge \bar{f}(\sigma) = a \wedge \bar{f}(\sigma a \sigma'') = a) = 0 \end{aligned}$$

This proves that $P(\Lambda^{(k)}_{f,M} < L) = 0$.

- For the probability that the period at time-point k is greater than U we have:

$$P(\Lambda^{(k)}_{f,M} > U) = \sum_{a \in A} P(\bar{f}(X^{(k)}_{f,M}) = a \ \wedge \bar{f}(X^{(k+1)}_{f,M}) \neq a \wedge \ldots \wedge \bar{f}(X^{(k+U)}_{f,M}) \neq a)$$

  Consider a path $\sigma \in (S \times A)^k \times S$ of length k, an arbitrary fixed action $a \in A$ and the path $\sigma' = \sigma a \sigma''$ where $\sigma'' \in (S \times A)^{U-1} \times S$ has length $U-1$ and $a \notin \sigma''[A]$. Note that path σ' has length $k+U$, and $a \notin A^{(U-1)}_{\sigma'}$.

  Since $a \notin A^{(U-1)}_{\sigma'}$ we have that $|A \setminus A^{(U-1)}_{\sigma'}| \geq 1$. For the case that $|A \setminus A^{(U-1)}_{\sigma'}| = 1$ we must have that $A \setminus A^{(U-1)}_{\sigma'} = \{a\}$ and then $F(\sigma') = \{a\}$ by the first case of Definition 7. It follows that in this case the scheduler f must select a with probability one. For the case $|A \setminus A^{(U-1)}_{\sigma'}| > 1$ we have, by Lemma 5, that $P(X^{(k+U)}_{f,M} = \sigma') = 0$.

  We have shown that either $P(X^{(k+U)}_{f,M} = \sigma') = 0$ or $P(\bar{f}(\sigma') \neq a) = 0$. For the probability that the period is greater than U we now find.

$$\begin{aligned} &P(\Lambda^{(k)}_{f,M} > U) = \sum_{a \in A} P(f(X^{(k)}_{f,M}) = a \ \wedge \bar{f}(X^{(k+1)}_{f,M}) \neq a \wedge \ldots \wedge \bar{f}(X^{(k+U)}_{f,M}) \neq a) \\ &\quad = \sum_{a \in A} \sum_{\sigma \in Paths^{(k)}_M} \sum_{\sigma'' \in Paths^{(U-1)}_M} P(X^{(k)}_{f,M} = \sigma \\ &\qquad \wedge X^{(k+U)}_{f,M} = \sigma a \sigma'' \wedge \bar{f}(\sigma) = a \wedge a \notin A^{(U-1)}_{\sigma''} \wedge \bar{f}(\sigma a \sigma'') \neq a) = 0 \end{aligned}$$

  This proves that $P(\Lambda^{(k)}_{f,M} > U) = 0$ and then that for all schedulers that are $[L, U]$ bounded fair we have that their period always lies between L and U.

- We now prove that for any scheduler that is $[L, U]$ bounded fair and any action $a \in A$ we have that the first occurrence of a is greater than U with probability zero. For this probability we have that:

$$P(\Delta^{(a)}_{f,M} > U) = P\big(\bar{f}(X^{(0)}_{f,M}) \neq a \wedge \ldots \wedge \bar{f}(X^{(U-1)}_{f,M}) \neq a\big).$$

  Consider a path σ of length $U-1$ such that the action a has not occurred yet in σ: $a \in A \setminus A^{(U-1)}_{\sigma}$. We will now show that either σ has probability zero to be reached by X or action a is selected by f with probability one. Since $A^{(U-1)}_{\sigma}$ does not contain a we have that $|A \setminus A^{(U-1)}_{\sigma}| \geq 1$. From Lemma 5 we now have that either $|A \setminus A^{(U-1)}_{\sigma}| = 1$ or $P(X^{(U-1)}_{f,M} = \sigma) = 0$. In the former case we find that $F(\sigma) = A \setminus A^{(U-1)}_{\sigma} = \{a\}$ and then

$P(\bar{f}(\sigma) = a) = 1$. For the probability that the first occurrence of a is greater than U we now find:

$$\begin{aligned} &P(\Delta^{(a)}_{f,M} > U) = \\ &\quad P(\bar{f}(X^{(0)}_{f,M}) \neq a \wedge \ldots \wedge \bar{f}(X^{(U-1)}_{f,M}) \neq a) = \\ &\quad \sum_{\sigma \in Paths^{(U-1)}_M} P(X^{(U-1)}_{f,M} = \sigma \ \wedge a \notin A^{(U-1)}_\sigma \wedge \bar{f}(\sigma) \neq a) = 0, \end{aligned}$$

since from $a \notin A^{(U-1)}_\sigma$ we have derived that either $P(X^{(U-1)}_{f,M} = \sigma)$ or $P(\bar{f}(\sigma) \neq a)$ is zero.

Bounded periods must be induced by bounded fair schedulers. We now prove that any scheduler for which the period lies between L and U with probability one and the first occurrence of any action is smaller or equal to U with probability one is $[L, U]$ bounded fair by contradiction. Assume f is a scheduler of M such that for any time-point $k \in \mathbb{Z}_{\geq 0}$ we have $P(\Lambda^{(k)}_{f,M} \in [L, U]) = 1$, for any action $a \in A$ we have that $P(\Delta^{(a)}_{f,M} \leq U) = 1$, and for which we have that f is not $[L, U]$ bounded fair. The latter means that there exists some path $\sigma \in (S \times A)^* \times S$ of length k and an action $a \in A$ such that $a \notin F(\sigma)$ and $P(\bar{f}(\sigma) = a) > 0$, i.e., f chooses an action which is not allowed according to the characteristic function F with probability greater than zero. We now show that this always leads to a contradiction.

We go over the different possibilities for $a \notin F(\sigma)$ given an arbitrary path σ of length k, with $P(X^{(k)}_{f,M} = \sigma) > 0$ and an arbitrary action a.

- First consider the case that $k \geq U$ and $|A \setminus A^{(U-1)}_\sigma| = 1$ and $a \in A^{(U-1)}_\sigma$. Obviously, since $F(\sigma) = A \setminus A^{(U-1)}_\sigma$ we have $a \notin F(\sigma)$. Now $|A \setminus A^{(U-1)}_\sigma| = 1$ means that there is exactly one action $b \in A$ which did not occur in the last $U - 1$ steps. There are now two possibilities: Either b occurred at an earlier time-point $i < k - U$ in the path σ or it did not yet occur.
  - In the first case we have that $P(\Lambda^{(i)}_{f,M} > U) > 0$ which is a contradiction with the fact that the period should be less than or equal to U.
  - For the second possibility we have that the first occurrence of b is at least $U + 1$ with probability one, which is a contradiction with the fact that the first occurrence should be less than or equal to U.
- Consider now the case that $k < L$ and $a \in A^{(k)}_\sigma$. Then there exists some earlier time-point $i < k$ where the action a occurred and for this time-point we find $P(\Lambda^{(i)}_{f,M} = k - i) > 0$. Since $k < L$ this is a contradiction with the fact that the period must be greater than or equal to L.
- Consider the case that $k < U$, $|A \setminus A^{(k)}_\sigma| = U - k$, and $a \in A^{(k)}_\sigma$. Then we have that there exists a path $\sigma a s$ such that $|A \setminus A^{(k+1)}_{\sigma a s}| = U - k$. This means that for one of the actions in $A \setminus A^{(k)}_\sigma$ we will have that it will not be scheduled for for another $U - k$ steps which means that for this action we find a period of at least $U - 1$. This is a contradiction with the fact that the period must be smaller than or equal to U.
- Finally we consider the case that $a \in A^{(L-1)}_\sigma$ then there exists some earlier time-point $k - L + 1 \leq i < k$ where the action a occurred and for this time-point we find $P(\Lambda^{(i)}_{f,M} = k - i) > 0$. Since $k - i < L$ this is again a contradiction with the fact that the period must be greater than or equal to L.

$\square$